JOURNAL OF COMPUTER AND SYSTEM SCIENCES 20, 50–58 (1980)

On Finding Minimal Length Superstrings

JOHN GALLANT

Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey 08540

DAVID MAIER*

Department of Computer Science, State University of New York at Stony Brook, Stony Brook, New York 11790

AND

JAMES A. STORER†

Bell Laboratories, Murray Hill, New Jersey 07974

Received February 21, 1978; revised August 17, 1979

A superstring of a set of strings $\{s_1, \ldots, s_n\}$ is a string s containing each s_i, $1 \leqslant i \leqslant n$, as a substring. The superstring problem is: Given a set S of strings and a positive integer K, does S have a superstring of length K? The superstring problem has applications to data storage; specifically, data compression. We consider the complexity of the superstring problem. *NP*-completeness results dealing with sets of strings over both finite and infinite alphabets are presented. Also, for a restricted version of the superstring problem, a linear time algorithm is given.

1. INTRODUCTION

DEFINITION 1. A *superstring* of a set of strings $S = \{s_1, \ldots, s_n\}$ is a string s containing each s_i, $1 \leqslant i \leqslant n$, as a substring. The *superstring problem* is: Given a set of strings S and a positive integer K, does S have a superstring of length K? We lose no generality by defining S to be a set, because if S is a collection of strings where some strings appear more than once, then S has exactly the same set of superstrings as the set $S' = \{s: s \text{ is in } S\}$. ∎

The superstring problem has applications to string storage. For example, in many programming languages, a character string may be represented by a pointer to that

* Research supported in part by NSF Grant DCR 74-21939 and in part by an IBM research fellowship.

† Research supported in part by NSF Grant DCR 74-21939 and in part by Bell Laboratories.

0022-0000/80/010050–09$02.00/0

string. The problem for the compiler is to arrange strings so that they may be "overlapped" as much as possible. The superstring problem also has more general applications to data compression as discussed in Storer (1977) and Storer and Szymanski (1977).

In the next section, we consider the superstring problem when the alphabet over which strings are written may be of arbitrary size. In Section 3, we consider fixed size alphabets; in particular, binary alphabets. Before proceeding to Section 2, we define the following notation:

If s and s_i denote strings and n is a positive 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 $\prod_{i=1}^{n} s$, with s^0 denoting the empty string. We extend this notation to apply to sets of strings in the obvious fashion and if s is a set of strings, s^* denotes $\bigcup_{i=0}^{\infty} s^i$.

If s is a string, $| s |$ denotes the length (in characters) of s, and if s is a set, $| s |$ denotes the cardinality of s and $\| s \|$ denotes $\sum_{x \in s} | x |$.

If h is a real number, $\lceil h \rceil$ denotes the least integer greater than or equal to h.

For an integer $n \geqslant 0$, $\mathrm{LEN}_2(n)$ denotes the number of bits necessary to write n in binary.

A string is *primitive* if no character appears more than once.

Two strings x and y have an *overlap* of length k if there exist strings u, v, and w with $| v | = k$, such that $x = uv$ and $y = vw$.

$G = (V, E)$ denotes a graph[1] G with vertex set V and edge set E. If G is directed, then we say that G is *loosely connected* if the corresponding undirected graph is connected. Also, for an undirected graph G with $v \in V$, we let $\mathrm{IN}(v)$ denote the number of incoming edges to v and $\mathrm{OUT}(v)$ denote the number of outgoing edges from v.

2. Unbounded Size Alphabets

In this section we consider superstring problems S, K, where no bound is assumed on the size of the alphabet over which S is written. Our first theorem shows the superstring problem to be *NP*-complete[2] even if for any integer $H \geqslant 3$, the restriction is made that all strings in the set must be primitive and of length H. This is a useful result in itself and, in addition, it makes the superstring problem an attractive *NP*-complete problem to be used to show other problems *NP*-complete. The proof of Theorem 1 employs a reduction involving the Hamilton path problem. For the case $H \geqslant 8$, a reduction employing a restricted version of the node cover problem appears in Maier and Storer (1977).

Before proceeding to Theorem 1, we provide a definition and a technical lemma.

Definition 2. The *directed Hamilton path (circuit) problem* is: Given a directed graph G, is there a path (cycle) that goes through each node of G exactly once. This

[1] Note that undirected graphs are always assumed to contain no self-loops. For a definition of graphs and related terms see Harary (1972).

[2] We show *NP*-completeness in the sense of Karp (1972) (which implies that of Cook (1971)). For a definition of *NP*-completeness and related terms see Aho *et al.* (1976).

problem is shown *NP*-complete in Karp (1972). The *restricted directed Hamilton path problem* is the directed Hamilton path problem with the following restrictions:

(1) There is a designated start node s and a designated end node t, with $\text{IN}(s) = \text{OUT}(t) = 0$.

(2) Except for the end node t, all nodes have out-degree greater than 1 ▮

LEMMA 1. *The restricted directed Hamilton path problem is NP-complete.*

Proof. Let G be an instance of the directed Hamilton circuit problem. Assume that G is connected, otherwise, no circuits exist. Now form the graph G' as follows. For condition (1), choose any vertex in G and split it into two nodes s and t, with s having all the outgoing edges and t having all the incoming edges. Next, for condition (2), add the new nodes a, b, and t', where t' is the new end node. Add an edge from all nodes with out-degree <2 to t' and add the edges (t, a), (t, b), (a, b), (b, a), (a, t'), and (b, t'). The reader can now check that G has a Hamilton circuit if and only if G' has a Hamilton path starting at s and ending at t. ▮

It should be noted that condition (1) is needed for the reduction in Theorem 1, but that condition (2) is only added to simplify the proof.

THEOREM 1. *The superstring problem is NP-complete. Furthermore, this problem is NP-complete even if for any integer $H \geqslant 3$, the restriction is made that all strings in the set be primitive and of length H.*

Proof. We first prove the theorem for nonprimitive strings of length 3 and then show how to modify the construction to make all strings primitive and of length H, for $H \geqslant 3$. Let $G = (V, E)$ be an instance of the restricted directed Hamilton path problem where V is the set of integers from 1 to n (1 is the start node and n the end node) and $|E| = m$. We construct strings for G over the alphabet $\Sigma = V \cup B \cup S$, where $B = \{\bar{v} \mid v \in V - \{n\}\}$ is the set of barred symbols, and $S = \{¢, \#, \$\}$ is the set of special symbols. In the reduction, the barred symbols may be thought of as local to a node while the unbarred symbols from V are thought of as global to the whole graph G.

For each node $v \in V - \{n\}$ we create a set A_v of $2^*\text{OUT}(v)$ strings. Let $R_v = \{w_0, \ldots, w_{\text{OUT}(v)-1}\}$ be the set of nodes adjacent to v. Then, $A_v = \{\bar{v}w_i\bar{v} \mid w_i \in R_v\} \cup \{w_i\bar{v}w_{i\oplus 1} \mid w_i \in R_v\}$, where $\oplus$ denotes addition modulo $\text{OUT}(v)$. For each node $v \in V - \{1, n\}$ create a singleton set C_v containing a string of the form $v \# \bar{v}$ called a *connector*. Finally, create a set T that contains elements called *terminal strings*: $T = \{¢ \# \bar{1}, n \# \$\}$.

Let S be the union of A_j, $1 \leqslant j < n$, C_i, $1 \leqslant i < n$, and T. We claim that G has a directed Hamilton path if and only if S has a superstring of length $2m + 3n$.

Suppose G has a directed Hamilton path. Let (v, w_i) be an edge on the path. First, create a superstring of length $2(\text{OUT}(v)) + 2$ for A_v of the form $\bar{v}w_i\bar{v}w_{i\oplus 1}\bar{v} \cdots \bar{v}w_i$,

called the *w_i-standard superstring* for A_v. This superstring is formed by overlapping the strings of A_v in the order

$$\bar{v}w_i\bar{v},\quad w_i\bar{v}w_{i\oplus 1},\quad \bar{v}w_{i\oplus 1}\bar{v},\quad \ldots,\quad \bar{v}w_{i\oplus \mathrm{OUT}(v)}\bar{v},\quad w_{i\oplus \mathrm{OUT}(v)}\bar{v}w_i$$

where each successive pair has an overlap of length 2. Note that the set of w_i-standard superstrings for A_v is in one-to-one correspondence with the cyclic permutations of the integers 0 through OUT(v) − 1. Also note that the standard superstrings for A_v are the only ones this short.

Let $(u_1, u_2, \ldots, u_n)$ denote the directed Hamilton path where $u_1 = 1$ and $u_n = n$, and abbreviate the u_j-standard superstring for A_{u_i} as STD($\bar{u}_i, u_j$). We can form a superstring for S by overlapping the standard superstrings and the strings in S but not in any A_v in the order:

$$¢ \# \bar{1}, \mathrm{STD}(\bar{1}, u_2), u_2 \# \bar{u}_2, \mathrm{STD}(\bar{u}_2, u_3), u_3 \# \bar{u}_3, \ldots, u_{n-1} \# \bar{u}_{n-1}, \mathrm{STD}(\bar{u}_{n-1}, n), n \# \$$$

This superstring has length $\sum_{i=1}^{n-1}(2{*}\mathrm{OUT}(i) + 2) + (n-2) + 4 = 2m + 3n$. The terms are for the standard superstrings, the #'s from the connectors, and the additional symbols from the terminal strings, respectively.

To prove the converse, we show that $2m + 3n$ is a lower bound on the size of a superstring for S and then show that this lower bound can only be achieved if the superstring encodes a directed Hamilton path.

There are a total of $2m + n$ strings, with a total length of $3(2m + n)$. The greatest amount of compression would result from an ordering in which each string except the first and last had an overlap of length 2 on both sides. This order would give a superstring of length $3(2m + n) - 2(2m + n - 1) = 2m + n + 2$. However, the $n - 2$ connectors can only have overlaps of length 1 on either side, since no string begins or ends with #. In addition, the terminal strings can overlap at most one symbol on only one side. Observing these requirements, we get a lower bound of $(2m + n + 2) + 2(n - 2) + 2 = 2m + 3n$ on the length of a superstring for S. Note that such a superstring must begin with $¢ \# \bar{1}$ and end with $n \# \$$.

Consider two consecutive occurrences of # in such a superstring. Let x be the string between the two #'s. The first symbol of x must be barred, and the last unbarred, since they are substrings of connectors. Since there are no connectors in x, all substrings of x except the first and the last must have overlaps of length two on both sides. The first string must be $\bar{v}u_j\bar{v}$, the next $u_j\bar{v}u_{j\oplus 1}$, and so on. Furthermore, all strings in A_v except two must have overlaps of length 2 on both sides, so every string in A_v but one must be succeeded in order by the unique string that overlaps it by 2. Thus all strings in A_v must occur contiguously in order, and since x contains one string from A_v, it must contain them all. Thus, x is the w_j-standard superstring for A_v.

By applying the above analysis to all sequential pairs of #'s we obtain $n - 1$ different standard strings. We can recover a directed Hamilton path by looking at the symbols next to each #, since the barred and unbarred symbol of each connector correspond to the same node in G. Note that by the location of $¢ \# \bar{1}$ and $n \# \$$, the path is from node 1 to node n.

We now address the restriction that all strings be primitive and of exactly length H,

for $H \geqslant 3$, by showing how to make the proper modifications on the strings in S. The alphabet Σ is augmented to include $\{\hat{a} \mid a \in V\}$. For $H = 3$, we need only change A_v. Replace strings of the form $\bar{v}a\bar{v}$ by the strings $\bar{v}a\hat{v}$, $a\hat{v}\hat{a}$, and $\hat{v}\hat{a}\bar{v}$. In addition, replace strings of the form $a\bar{v}b$ by $\hat{a}\bar{v}b$. For $H \geqslant 4$ let y and y' be primitive strings over an alphabet disjoint from Σ of length $H - 4$ and $H - 2$ respectively. Replace the # in all connectors and terminals by y'. For A_v, replace strings of the form $\bar{v}a\bar{v}$ by $\bar{v}ay\hat{a}\hat{v}$ and those of the form $a\bar{v}b$ by $\hat{a}\hat{v}y\bar{v}b$. We leave it to the reader to verify that with these changes there is an integer k such that the theorem holds. The proofs differ only slightly from the one for the case of nonprimitive strings of length 3. The reader can also check that the superstring problem is in *NP* and that the above reductions can be done in polynomial time. ∎

In view of the last theorem, it is natural to consider what happens if all strings are of length shorter than 3. The next theorem and its corollary present a linear time algorithm to find a minimal length superstring for a set of strings of length less than or equal to 2. Before proceeding to Theorem 2, we have a definition and a lemma that are used in the proof.

DEFINITION 3. For a directed graph $G = (V, E)$, if $G_1 = (V_1, E_1), \ldots, G_k = (V_k, E_k)$ are the loosely connected components of G, then let

$$\text{PATH}(G) = \sum_{i=1}^{k} \max\left\{1, \sum_{v \in V_i} \frac{|\text{IN}(v) - \text{OUT}(v)|}{2}\right\}.$$

A *path-decomposition* of G is a partition of E into edge disjoint paths. ∎

LEMMA 2. *The number of paths in a minimal path-decomposition of a directed graph G is given by* PATH(G).

Proof. Algorithm 1 produces a minimal path decomposition P for a directed graph $G = (V, E)$. Each time a path p is deleted from G (and added to P) in the first *WHILE* loop, the outdegree of the start node of p is reduced by 1, the indegree of the end node of p is reduced by 1, and all other nodes v of p have both IN(v) and OUT(v) reduced by 1 (i.e., $|\text{IN}(v) - \text{OUT}(v)|$ is unchanged). Hence, this loop produces $\sum_{v \in V} |\text{IN}(v) - \text{OUT}(v)|/2$ paths. The second *WHILE* loop adds a new path to P only when a loosely connected component, consisting entirely of cycles (i.e., IN(v) = OUT(v) for all nodes v in this component), is encountered for the first time. ∎

WHILE there exists a node v in G with IN(v) < OUT(v) *DO*
    Starting at v, traverse edges at random until a node with no outgoing edges is reached, delete the edges traversed from G, and add this path to P.
WHILE G is not empty *DO*
    IF there exists a cycle c which intersects a path p in P
        THEN Delete c from G and "splice" it into p.
        ELSE Delete a cycle from G and add it to P.

Algorithm 1

THEOREM 2. *For a set of strings* $S = \{s_1, \ldots, s_n\}$ *and an integer* K, *if* $| w_i | \leqslant 2$, $1 \leqslant i \leqslant n$, *then there is a linear time and space algorithm (on a* RAM[3]*) to decide if* S *has a superstring of length* K.

Proof. Let Σ denote the alphabet over which S is written. We can assume that all strings in S have length exactly 2 because strings of length 1 are either a substring of a string of length 2 or are a unique character not appearing anywhere else in S. We can also assume all strings in W to be primitive since for a nonprimitive string $s_i = aa$ ($a \in \Sigma$) in S, if the character a does not appear anywhere else in S, then S has a superstring of length K if and only if $S - \{s_i\}$ has a superstring of length $K - 2$; otherwise, S has a superstring of length K if and only if $S - \{s_i\}$ has a superstring of length $K - 1$. We can associate a directed graph $G = (V, E)$ with S by letting $V = \Sigma$ and $(a, b) \in E$ when $ab \in S$. The reader can now verify that S has a superstring of length K if and only if $\text{PATH}(G) \leqslant K - | S |$ and that $\text{PATH}(G)$ can be computed using linear time and space. ∎

COROLLARY 2.1. *There is a linear time and space algorithm to find a minimal length superstring for a set of strings of length less than or equal to* 2.

Proof. A single pass over S suffices to make a list of all the distinct characters in S and the number of times each character appears in strings of length 1 and strings of length 2. In a second pass over S, all strings of length 1 can be deleted from S and the strings of length 1 that are not a substring of any other string in S can be saved and concatenated on to the superstring produced for the remainder of S. Thus, in view of the proof of Theorem 2, it suffices to show a linear time and space algorithm to find a minimal size path-decomposition P for a directed graph G. An algorithm for this problem was presented in Lemma 2. Using linked lists, this algorithm may be implemented in linear time. ∎

COROLLARY 2.2. *For a multiset*[4] *of strings* S *over alphabet* Σ, *algorithms exist to find a minimal length superstring for* S *which use the following amounts of time and space*:

(1) *Linear expected time and linear space.*

(2) $o(\| S \| \text{LEN}_2 | S |)$ *time and linear space.*

(3) *Linear time and* $o(| S | + | \Sigma |^2)$ *space.*

Proof. An algorithm is easily constructed for (1) using hashing techniques and for (2) using dictionary techniques. For (3), strings of length 1 can be dealt with as in Corollary 2.1 and the number of times each string of length 2 occurs may be tabulated in linear time by using an $o(| \Sigma |)$ by $o(| \Sigma |)$ matrix. Note that this matrix may be

[3] For a discussion of the RAM (random access machine) model and related terms, see Aho *et al.* (1976). Also, note that whenever we refer to time or space complexity, we shall always be referring to worst case complexity unless otherwise stated.

[4] The concept of a multiset is similar to that of a set except that repetitions are allowed. For example, $\{a, a, b\}$ is a multiset. Note that a set is a special case of a multiset.

effectively initialized to all zeros in linear time by using a well-known technique which employs an $o(|W|)$ stack and a "hand shaking" protocol. ▮

Theorem 2 and its corollaries have several practical applications. One is for storing Huffman trees[5] for encoding letter pairs. Another is for storing a directed graph G. The path decomposition of G can be computed and then the superstrings of the paths, separated by markers, can be stored. If d denotes the amount of space to store a node name, this scheme requires space $d(|E| + 2\text{PATH}(G) - 1)$. This compares favorably with other methods such as storing a list of edges which takes space $2d|E|$ or storing an adjacency list which takes space $d(|E| + 2|V| - 1)$.

3. Bounded Size Alphabets

We shall let Σ denote the alphabet over which strings are written. In the last section no bound was assumed on $|\Sigma|$. This models many situations such as those in which entries in a system dictionary are taken to be the basic "characters." However, there are certainly many situations where $|\Sigma|$ is assumed to be some fixed size such as 256 or, perhaps, 2. It is easy to see that the superstring problem remains *NP*-complete when restricted to a three-symbol alphabet. We can take an alphabet $\Sigma = \{a_1, ..., a_m\}$ and encode a_i, $1 \leqslant i \leqslant m$, over the alphabet $\Sigma' = \{0, 1, a\}$ by writing a_i as $\bar{i}a$ where $\bar{i}$ denotes i written in binary using $\text{LEN}_2(m)$ bits (padding to the left with zeros when necessary). We now see that the set $S = \{s_1, ..., s_n\}$ written over the alphabet Σ has a superstring of length K if and only if S written over the alphabet Σ' has a superstring of length $(\text{LEN}_2(m) + 1)K$. With a little more thought, one can also see how to encode Σ over the alphabet $\{0, 1\}$. We show a stronger result in this section. Theorem 3 states that the superstring problem over the alphabet $\Sigma = \{0, 1\}$ is *NP*-complete even if for any real $h > 1$, the restriction is made that all strings in the set S must be of length $\lceil h \text{LEN}_2 \| S \| \rceil$. The reader can check that if $h < 1$, it is not possible for a set $S = \{s_1, ..., s_n\}$ to have $|s_i| = \lceil h \text{LEN}_2 \| S \| \rceil$, $1 \leqslant i \leqslant n$, (provided $\| S \|$ is sufficiently large) and hence our result is, in a sense, almost as strong as possible (i.e., the only case we have left out is that of $h = 1$).

Theorem 3. *The superstring problem is NP-complete even if for any real number $h > 1$, the problem is restricted to instances S, K where S is written over the alphabet $\{0, 1\}$ and all strings in S have length $\lceil h \text{LEN}_2 \| S \| \rceil$.*

Proof. Due to both the construction used in Maier and Storer (1977, Theorem 1) and that used in our Theorem 1, we can assume that all strings in an instance of the superstring problem have length 8 but can be padded to any length. In other words, for $i \geqslant 8$, we can define a series of sets $S_i = \{s_{i,1}, ..., s_{i,n}\}$, where:

(1) $|s_{i,j}| = i$, $1 \leqslant j \leqslant n$.

[5] For a discussion of Huffman trees, see Huffman (1952).

(2) S_8 has a superstring of length K if and only if S_i has a superstring of length $K + n(i - 8)$.

(3) If we let $\Sigma = \{a_1, ..., a_m\}$ denote the alphabet over which S_8 is written, then S_i is also written over Σ. Note that we are not insisting that the strings in S_i be primitive and the proof of Theorem 1 must be modified to make this condition hold (two \$'s in the same string must denote the same character).

For nonnegative integers i and j such that $\mathrm{LEN}_2(i) \leqslant j$ let $\bar{i}_j$ denote i written in binary using j bits (padding to the left with zeros when necessary). Let $J = \mathrm{LEN}_2 \| S_8 \| + 2$ and for $1 \leqslant i \leqslant m$ let

$$a_i' = (0^J)(1\bar{i}_{J-2}0)(1^J).$$

For $8 \leqslant i$ and $1 \leqslant j \leqslant n$, let $s'_{i,j}$ be $s_{i,j}$ with a_k', $1 \leqslant k \leqslant m$, substituted for all occurrences of a_k in $s_{i,j}$ (note that $| s'_{i,j} | = 3Ji$). Now let H be the least integer greater than or equal to 8 such that $3JH \geqslant \lceil h \, \mathrm{LEN}_2(3J \| S_H \|) \rceil$ and $H \leqslant h(H - 1)$ and let

$$X = \{s'_{H,j} : 1 \leqslant j \leqslant n\},$$

$$Y = \left\{ \bigcup_{i=0}^{L} f(0^{J+1}\bar{i}_{3JH-2(J+1)}1^{J+1}) \right\},$$

$$S = X \cup Y,$$

where f is the function from strings to sets of strings defined by

$$\begin{aligned} f(x) &= \{\ \} && \text{if } x \in \{0\}^{J+1}\{0, 1\}^*\{0^{J+1}, 1^{J+1}\}\{0, 1\}^*1^{J+1} \\ &= \{x\} && \text{otherwise} \end{aligned}$$

and L is the least integer which causes S to satisfy $\lceil h \, \mathrm{LEN}_2 \| S \| \rceil = 3JH$. We must verify that S is well defined. First, let us see how large $\lceil h \, \mathrm{LEN}_2 \| S \| \rceil$ would be if we let Y be as small as possible in the definition of S (i.e., $Y = \{\ \}$). We see:

$$\lceil h \, \mathrm{LEN}_2 \| S \| \rceil = \lceil h \, \mathrm{LEN}_2(3J \| S_H \|) \rceil \leqslant 3JH.$$

Now let us see how large $\lceil h \, \mathrm{LEN}_2 \| S \| \rceil$ would be if we let Y be as large as possible in the definition of S (i.e., $L = 2^{3JH-2(J+1)} - 1$). An upper bound on how many strings are eliminated in the definition of S by the function f is $2(3JH - 3(J + 1) + 1)(2^{3JH-3(J+1)})$. We see

$$\begin{aligned} &2^{3JH-2(J+1)} - 2(3JH - 3(J + 1) + 1)(2^{3JH-3(J+1)}) - 1 \\ &\quad = (2^{3JH}/2^{3(J+1)})(2^{J+1} + 6(J + 1) - 6JH + 2) - 1 \\ &\quad > (2^{3JH})/(2^{3(J+1)}) \end{aligned}$$

and so we see:

$$\begin{aligned} \lceil h \, \mathrm{LEN}_2 \| S \| \rceil &\geqslant \lceil h \, \mathrm{LEN}_2(3JHn + 3JH(2^{3JH})/(2^{3(J+1)})) \rceil \\ &\geqslant \lceil 3hJ(H - 1) \rceil \\ &\geqslant 3JH. \end{aligned}$$

Thus, since each string in Y has length less than $|X|$, it follows that S is well defined (i.e., each time a string is added to Y, $\text{LEN}_2 \| S \|$ increases by at most 1). At this point the reader should also verify that $\| S \| \leqslant u(\| S_8 \|)$ for some polynomial u depending only on h.

We see that all strings in the set S have length $\lceil h \text{ LEN}_2 \| S \| \rceil$, start with a 0, and end with a 1. Thus, since all strings in Y start with $J + 1$ 0's and end with $J + 1$ 1's and no string in Y contains $J + 1$ 0's or $J + 1$ 1's as a proper substring, the strings in Y cannot overlap with each other. Also, no strings in Y can overlap with a string in X since no string in X contains $J + 1$ 0's or $J + 1$ 1's as a (not necessarily proper) substring. The reader can now verify that if two strings in X overlap with each other then they must do so in multiples of $3J$ characters and so S_8 has a superstring of length K if and only if S has a superstring of length $3J(K + n(H - 8)) + \| Y \|$. Also, the above reduction can be done in polynomial time. ∎

4. Conclusion

Because the superstring problem has many practical applications, the *NP*-completeness results presented in this paper should not discourage future research regarding the superstring problem. Rather, they should provide the impetus for studying approximation algorithms and heuristics for finding a minimal length superstring.

References

A. V. Aho, J. E. Hopcroft, and J. D. Ullman (1976). "The Design and Analysis of Computer Algorithms," 2nd printing, Addison–Wesley, Reading, Mass.

S. A. Cook (1971). The complexity of theorem proving procedures, *in* "Proceedings, Third Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio," pp. 151–158.

M. R. Garey, D. S. Johnson, and L. Stockmeyer (1976). Some simplified *NP*-complete problems, *Theor. Comp. Sci.* **1**, 237–267.

F. Harary (1972). "Graph Theory," Addison–Wesley, Reading, Mass.

D. A. Huffman (1952). A Method for the Construction of Minimum-Redundancy Codes, *Proc. IRE* **40**, 1098–1101.

R. M. Karp (1972). Reducibility among combinatorial problems, *in* "Complexity of Computer Computations" (R. E. Miller and J. W. Thatcher, Eds.), pp. 85–103, Plenum, New York.

D. Maier and J. A. Storer (1977). "A Note on the Complexity of the Superstring Problem," Technical Report 233, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, N.J.

J. A. Storer (1977). "*NP*-Completeness Results Concerning Data Compression," Technical Report 234, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, N.J.

J. A. Storer and T. G. Szymanski (1977). "The Macro Model for Data Compression," Technical Report 235, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, N.J.