S. P. Ghosh
M. E. Senko

String Path Search Procedures for Data Base Systems

Abstract: This paper structures algorithms for the translation of set theoretic queries into procedures for the search of arbitrary complex networks constructed on a data base using three basic types of strings. A method for parametrization of queries which is appropriate for accessing string structures is outlined and it is shown how the properties of string structures can be used to construct an algorithm for finding a search path with minimum path cardinality for a given query addressed to such a network. (The term data management system is used instead of data base management system.)

1. Introduction

In a computing environment, data management systems deal with the problems of storing and retrieving information from large complex data bases based on non-numerical types of information. Like any other software system, a data management system has a high-level language, called a query language for user interaction with the data. The primitives of a query language are based on a data description. Usually the data bases are very large; hence; different kinds of data accessing rules are also associated with data management systems for efficient retrieval of subsets of data requested by different queries. Some data management systems have more than these two levels contained in them. A good review of many different data management systems is given in the CODASYL-system committee's technical report [1].

In the last few years there has been a trend to introduce multiple independent levels in data management systems. In this approach a user can interact with the system at any level without having to know the levels that are below it (i.e., closer to the hardware). A good example of work on this subject is reflected in the Data Independent Accessing Model (DIAM) system architecture developed by Senko et al. [2]. In the DIAM the data management system is divided into four levels, viz., data description model, access structure model, encoding model, and physical model. The data description model is based on an entity set concept. A query language based on entity sets has been developed by Fehder [3]. The access structure model is based on string structures which define access paths through different subsets of the data. A neat presentation of string structures has been given by Altman et al. [4].

When a user is free to request queries without knowledge of the access structures embedded in the data base, the computer has to take the responsibility of analyzing the query and reducing it to a parametric form such that the parameters can be used to select the proper access paths for retrieving the subsets of the data relevant to the query. One of the purposes of this paper is to investigate the methods for parametrizing the queries based on entity set concepts. The major part of the paper is devoted to studying properties of access path structures that can be constructed from string structures. The primary reason for studying properties of string structures is that these properties can be used to construct efficient algorithms for answering queries based on entity set concepts. Many authors have done basic work on search algorithms. Knuth [5] provides an excellent treatment on searching tree, multilist, and other structures. Theories and properties of complex queries and data structures involving paths between data units have not been discussed up to now. Based on the results of this paper Ghosh and Astrahan [6] and Astrahan and Ghosh [7] have developed detailed algorithms for selecting optimum search paths for providing answers to queries based on entity sets. This paper does not discuss optimum construction of string structures over a data base. It is assumed that the string structures necessary to answer a given query are available.

The materials presented in this paper are organized into six sections. The following section deals with string structure description of data. Section 3 contains a method of parametrizing queries to expedite a search through string structures. Section 4 deals with properties of search

paths. Section 5 provides a general algorithm for obtaining a search path with minimum path cardinality, and section 6 summarizes the results and provides a discussion.

2. String structure description

String structures were introduced by Senko et al. [2] and formalized by Altman et al. [4]. The reader may refer to these papers for a detailed description of the subject. For completeness of our paper a sketchy description of string structures is provided in this section.

A string is an access path through the data. There are three basic string types:

i. The A-string (atomic string) is used to characterize the access paths connecting sets and/or subsets of attribute domain name/role name/attribute values.
ii. The E-string (entity string) is used to characterize the access paths connecting structurally homogeneous (i.e., having the same type description) sets of subsets of data.
iii. The L-string (link string) is used to characterize the access paths connecting structurally heterogeneous sets and/or subsets of data.

Each string type is defined in a generic form over attribute values or a set of string types which are referred to as the components and are specified in the exiting list (*EXL*) of the string type definition format. A string type can have multiple instances in a data base. The string type definition format also contains a name, its kind, a string selection criterion and an *ON* criterion. An exiting list is a list of the components (i.e., strings or attributes) which are used to construct the instances of the particular string type. String criteria (SC, with values *SC*) can specify the criteria on the bases of which instances of the components are selected and/or ordered for constructing the instances of the particular string type. The ordering of the instances of the components can also be specified through the exiting list. The string criteria represent three concepts in the description given by Altman et al. [4], viz., selection criteria, matching criteria, and ORDER ON criteria. This substitution is done to improve parametrization of the string structures. The *ON* criterion specifies the names of the strings for which the particular string type is a component.

A string catalogue provides a list of the descriptions of all the string types defined on the data. Henceforth, for simplicity, the word "string" is used instead of "string type."

Example 1

Suppose we have a personnel file. Each person has children and holds a number of jobs. The attributes in this file are man number, man name, man age, child(ren) name(s), job name(s) and average wage(s). A catalogue of the string types for the data representing a man, his children, his jobs, and the corresponding associations is given as follows:

Person	Entity Set	[(P. Man No.)]
*PA*1	*ASG*	[*EXL* = (*P. Man No.*, *P. Man Name*, *P. Man Age*); *ON* = *PE*1]
*PE*1	*ESG*	[*EXL* = (*PA*1); 00 = (*P. Man No.*)]

P. Man No. is the name given to the attribute man number when it is a property of the entity person. Similarly, *P. Man Name* and *P. Man Age* are attributes of the person. In this catalogue each instance of the person entity set is identified by its *P. Man No.* The three attributes are components of the string *PA*1, which is an A-string (denoted by *ASG*) with *EXL* = (*P. Man No.*, *P. Man Name*, *P. Man Age*). The string *PE*1 is defined on *PA*1. An instance of *PA*1 is obtained by constructing an access path through one value of *P. Man No.*, one value of *P. Man Name*, and one value of *P. Man Age*, all belonging to the same person. The value of *P. Man No.* is the first element in an instance of *PA*1, the value of *P. Man Name* is the second element, and so on. *PE*1 is an E-string (denoted by *ESG*) constructed over *PA*1. It has one instance and is an access path connecting all the instances of *PA*1 in a sequential manner, ordered on the values of *P. Man No.* The entry 00 is referred to as the SC for *PE*1; *PE*1 connects only the heads of the list of *P. Man No.*, *P. Man Name*, and *P. Man Age*.

The following strings are constructed on the child and job entity sets.

Child	Entity Set	[(C. Man No., C. Child Name)]
*CA*1	*ASG*	[*EXL* = (*C. Man No.*, *C. Child Name*); *ON* = *CE*1]
*CE*1	*ESG*	[*EXL* = (*CA*1); 00 = (*C. Man No.*, *C. Child Name*)]
C. Child Name	*Attribute*	[*ON* = *CA*1]
C. Man No.	*Attribute*	[*ON* = *CA*1]
Job	Entity Set	[(J. Man No., J. Job Name, J. Av. Sal.)]
*JA*1	*ASG*	[*EXL* = (*J. Man No.*, *J. Av. Sal.*); *ON* = *JE*1]
*JE*1	*ESG*	[*EXL* = (*JA*1); 00 = (*J. Job Name*)]
*JA*2	*ASG*	[*EXL* = (*J. Man No.*); *ON* = *JL*1]
*JA*3	*ASG*	[*EXL* = (*J. Man No.*, *J. Job Name*, *J. Av. Sal.*); *ON* = *JE*2]

JE2	*ESG*	[*EXL* = (*JA3*); *PART* = *J. Man No.*); 00 = (*J. Job Name*); *ON* = *JL1*]
JL1	*LSG*	[*EXL* = (*JA2*, *JE2*); *MC* = (*J. Man No.* = *J. Man No.*); *ON* = *JE3*]
JE3	*ESG*	[*EXL* = (*JL1*); 00 = (*J. Man No.*)]
J. Av. Sal	*Attribute*	[*ON* = *JA1*; *ON* = *JA3*]
J. Job Name	*Attribute*	[*ON* = *JA1*; *ON* = *JA3*]
J. Man No.	*Attribute*	[*ON* = *JA1*; *ON* = *JA2*; *ON* = *JA3*].

In the string *CE1*, the SC is an ORDER ON criterion on the values of two attributes. It implies that the instances of *CA1* are connected in a sequential manner ordered on the values of *C. Man No.* For a fixed value of *C. Man No.*, the instances are ordered on the values of *C. Child Name*. *JE2* is an E-string with multiple instances, where each instance corresponds to a distinct value of *J. Man No.* Within each instance of *JE2* there may be multiple instances of *JA3*. The instances of *JA3* within each instance of *JE2* are ordered on the values of *J. Job Name*. *JL1* is an L-string (denoted by *LSG*) whose components are *JA2* and *JE2*. An instance of *JL1* is obtained by constructing an access path from an instance of *JA2* to an instance of *JE2* where both have the same value of *J. Man No.* In this catalogue *J. Man No.* is an attribute with three A-strings defined on it.

If this catalogue is critically examined, it is obvious that the L-string *JL1* is redundant. The instances of *JE3* may be used as entry points to the network thus described by the catalogue.

3. Parametrization of queries

Fehder [3] and others working with query languages have provided some ideas for parametrization of queries. It appears that the best method is to base the parametrization on the data description because the queries are used for extracting information from the data base. Senko et al. [2] used the data description as the basis for defining access path structures. When such structures are available they should be used for parametrization of queries so that identification of search paths relevant to a query are simplified. The approach taken in this section is based on this concept.

A query seeks some information from data that satisfy certain conditions. If the data base were described in terms of attributes, values, and entities, then the information which the data can supply can be translated into these parameters. It is assumed, for this paper, that there exists a translator which transforms the query from the query language form to the parametrized form. Thus, for the query which states "find the records of all employees who are over 40 years of age when hired" from a data base which does not contain "age when hired" but contains "date of birth" and "date of hire," the attribute "age when hired" has to be transformed into a function of two attributes, "date of birth" and "date of hire." Without such a translation a search cannot be performed on the data base.

In the parametrized form the attributes specified by the query can be classified into two groups, viz., qualification attributes and output attributes. In some cases, it is possible for an attribute to play a dual role. In our description the output attributes are those which have no restrictions imposed on their values by the queries but are relevant to the query. A qualification criterion (QC, with values *QC*) of a query is stated in terms of functions of qualification attributes and their values. The values, or some function of the values, of the output attributes in one or more entities which satisfy the QC are the information desired by the query. Thus, when seeking the records of all employees with "age at hire > 40," the QC attributes are "date of hire" and "date of birth." The values of these two attributes specified by the *QC* are their whole domain because corresponding to any value of one of the attributes, there can exist a value of the other attribute such that the instance of the entity set is relevant to the query. Here the output attributes of the query are all the attributes in the entity set "employee."

In a data base which is described by entities and their properties, it is easy to see that all subsets of the entity sets can be defined in terms of Boolean functions of attributes, where each attribute specifies subsets of its domain, i.e., are of the type

$$(A_1 = \theta_1) \wedge (A_2 = \theta_2) \vee (A_3 = \theta_3) \vee \cdots \wedge (A_l = \theta_l) \quad (1)$$

where $A_1, A_2, \cdots, A_l$ are the attributes and the θ_i are a subset of the Ω_i (the domain of A_i) $i = 1, 2, \cdots, l$. (As in most set theoretic situations it is assumed that a set, as well as the null set, can be regarded as a subset of itself.) Thus the SC of any string can be represented by functions and/or equations in functions of the type (1). A Boolean function of this type which contains only $\wedge$ (AND) functions, i.e.,

$$(A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge (A_3 = \theta_3) \wedge \cdots \wedge (A_l = \theta_l) \quad (2)$$

is called a canonical form or canonical term. It is easy to see that any general (well defined) Boolean function of the type (1) can be expanded into a number of canonical terms connected by $\vee$ (OR) functions. If an SC is an ORDER ON criterion on an attribute A_i, then $SC \equiv (A_i = \Omega_i)$. Other uses of ORDER ON criteria in search

are not discussed here. Thus, for parametrization it is sufficient to identify the QCs by Boolean functions.

In some situations the query may specify a complex function over some attributes, say, $A_1, A_2, \cdots, A_l$; then for retrieval purposes QC is of the form $(A_1 = \Omega_1) \wedge (A_2 = \Omega_2) \wedge \cdots \wedge (A_l = \Omega_l)$. For simplicity of analysis it is assumed that the SC and the QC are all expanded in canonical form. All the results in this paper are derived under the assumption that QC and SC contain only one canonical term each. If they contain more than one term in the canonical expansion, then the procedures described hereafter have to be repeated for all pairs of terms between SC and QC. To determine whether the access path defined by a string is relevant to a query or not, SC is compared with QC.

Let θ_i and θ_i' be two sets of values of A_i. If $\theta_i \cap \theta_i' = \emptyset$ (empty set), then θ_i negates θ_i'. It also implies that the two criteria $QC \equiv (A_i = \theta_i)$ and $SC \equiv (A_i = \theta_i')$ negate each other and there is no intersection between the SC and the QC. If $\theta_i \supseteq \theta_i'$ then (QC covers SC) $\equiv$ (QC contains SC) $\equiv$ (SC is contained in the QC), and there is an overlap between QC and SC. Similarly, the concept of the QC covering the SC can be defined when they contain more than one attribute. When a QC is compared with a SC the following situations may arise:

1. *The default SC (as in an A-string)* $\Rightarrow$ The relevance of the query to the string depends on the exiting list of the string. If the exiting list contains all the QC attributes and the output attributes, then the SC contains the answer for the QC; otherwise, the SC does not completely answer the QC. If the exiting list contains other strings, then the exiting lists of those strings, along with their SC, have to be examined against the QC.
2. *The SC attributes are identical with the QC attributes* $\Rightarrow$ The values of the attributes determine whether the QC is contained in the SC. If the values specified by at least one SC attribute negate the values of that QC attribute, then the SC negates the QC. Otherwise, there is an overlap between the SC and the QC. If the values specified by each QC attribute are contained in the values of the corresponding SC attribute. then the SC contains the QC.
3. *The SC attributes are a subset of the QC attributes* $\Rightarrow$ If the values specified by at most one SC attribute negate the values of that QC attribute, then the SC negates the QC. Otherwise, there is an overlap. If the values specified by each SC attribute contain the values of the corresponding QC attribute, then the SC contains the QC.
4. *The QC attributes are a subset of the SC attributes* $\Rightarrow$ If the values specified by at least one QC attribute are negated by the values of that SC attribute, then the SC negates the QC. Otherwise, there is an overlap. If the values specified by each QC attribute contain the values of the corresponding SC attribute, then the SC is contained in the QC.
5. *A subset of the SC attributes is contained in the set of QC attributes* $\Rightarrow$ If the values specified by at most one of the common attributes of SC negate the values of that QC attribute, then the SC negates the QC. Otherwise there is an overlap.

When comparing the QC with the SC, very often the QC will not be completely contained in the SC and thus only a subset of the information requested by the query can be retrieved from the corresponding string. The remaining portion of the information desired by the query has to be retrieved from other strings. Thus it is essential to obtain an analytic expression for the non-overlapping portion (of the QC) between a QC and a SC. Even when the SC and the QC are canonical terms, in general, the non-overlapping portion contains multiple terms. In the following analysis the non-overlapping portion is expressed as the union of disjoint canonical terms. One of the advantages of this type of disjoint canonical decomposition is that it eliminates redundant searching.

An attribute can specify a single value or a range of values or a set of discrete values. We use θ and ι to denote any of these three types; $\theta - \iota$ denotes the subset of θ when ι has been deleted. The criteria are assumed to be of the following form:

$$QC = (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l), \text{ and} \quad (3)$$

$$SC = (A_1 = \iota_1) \wedge (A_2 = \iota_2) \wedge \cdots \wedge (A_{l_1} = \iota_{l_1}). \quad (4)$$

• *SC and QC are based on the same attributes*
Consider only two attributes; then

$$\begin{aligned} QC - SC &= (A_1 = \theta_1) \wedge (A_2 = \theta_2) \\ &\quad - (A_1 = \iota_1) \wedge (A_2 = \iota_2) \\ &= ((A_1 = \theta_1) \wedge (A_2 = (\theta_2 - \iota_2))) \\ &\quad \vee ((A_1 = (\theta_1 - \iota_1)) \wedge (A_2 = (\iota_2 \cap \theta_2))). \end{aligned} \quad (5)$$

For three attributes,

$$\begin{aligned} QC - SC &= (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge (A_3 = \theta_3) \\ &\quad - (A_1 = \iota_1) \wedge (A_2 = \iota_2) \wedge (A_3 = \iota_3) \\ &= ((A_1 = (\theta_1 - \iota_1)) \wedge (A_2 = \theta_2) \\ &\quad \wedge (A_3 = \theta_3)) \vee ((A_1 = (\iota_1 \cap \theta_1)) \\ &\quad \wedge (A_2 = (\theta_2 - \iota_2)) \wedge (A_3 = \theta_3)) \\ &\quad \vee ((A_1 = (\iota_1 \cap \theta_1)) \wedge (A_2 = (\iota_2 \cap \theta_2)) \\ &\quad \wedge (A_3 = (\theta_3 - \iota_3))). \end{aligned} \quad (6)$$

Hence, in general,

$$QC - SC = \bigvee_{i=0}^{l-1} \Big(\bigwedge_{j=1}^{i} (A_j = (\iota_j \cap \theta_j)) \wedge (A_{i+1} = (\theta_{i+1} - \iota_{i+1})) \bigwedge_{j=i+2}^{l} (A_j = \theta_j) \Big). \quad (7)$$

Thus, when the QC and the SC are based on the same l attributes, the difference (same as non-overlap) $QC - SC$ can have a maximum of l canonical terms. If, for any attribute $\theta_j \equiv \iota_j$, then $A_j = (\theta_j - \iota_j)$ denotes the empty set and the cooresponding term drops off. The operation $\bigwedge_{j=1}^{0} \Rightarrow$ intersection of zero number of sets $\equiv$ whole space.

• *SC attributes are a subset of the QC attributes*

When the QC has two attributes and the SC has one attribute, then

$$QC - SC = (A_1 = \theta_1) \wedge (A_2 = \theta_2) - (A_1 = \iota_1) = (A_1 = (\theta_1 - \iota_1)) \wedge (A_2 = \theta_2). \quad (8)$$

When the QC has three attributes and the SC has two attributes, then

$$QC - SC = (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge (A_3 = \theta_3) - (A_1 = \iota_1) \wedge (A_2 = \iota_2) = ((A_1 = (\theta_1 - \iota_1)) \wedge (A_2 = \theta_2) \wedge (A_3 = \theta_3)) \vee ((A_1 = (\iota_1 \cap \theta_1)) \wedge (A_2 = (\theta_2 - \iota_2)) \wedge (A_3 = \theta_3)). \quad (9)$$

In general, when the QC has l attributes and the SC has l_1 attributes and $l_1 \leq l$, then

$$QC - SC = (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l) - (A_1 = \iota_1) \wedge (A_2 = \iota_2) \wedge \cdots \wedge (A_{l_1} = \iota_{l_1}) = \bigvee_{i=0}^{l_1-1} \Big(\bigwedge_{j=1}^{i} (A_j = (\iota_j \cap \theta_j)) \wedge (A_{i+1}) = (\theta_{i+1} - \iota_{i+1})) \bigwedge_{j=i+2}^{l} (A_j = \theta_j) \Big). \quad (10)$$

Thus, when $l_1 \leq l$, the maximum number of terms in the disjoint canonical decomposition of the non-overlapping portion of the QC with respect to the SC is l_1.

• *QC attributes are a subset of the SC attributes*

Suppose the QC has one attribute and the SC has two attributes; then

$$QC - SC = (A_1 = \theta_1) - (A_1 = \iota_1) \wedge (A_2 = \iota_2) = ((A_1 = (\theta_1 - \iota_1)) \vee (A_1 = (\iota_1 \cap \theta_1))) \wedge (A_2 = (\Omega_2 - \iota_2)). \quad (11)$$

If the QC has one attribute and the SC has $l_1 > 1$ attributes, then

$$QC - SC = (A_1 = \theta_1) - (A_1 = \iota_1) \wedge (A_2 = \iota_2) \wedge \cdots \wedge (A_{l_1} = \iota_{l_1}) = (A_1 = (\theta_1 - \iota_1)) \bigvee_{i=2}^{l_1} \Big(\bigwedge_{j=1}^{i-1} (A_j = (\iota_j \cap \theta_j)) \wedge (A_i = (\Omega_i - \iota_i)) \Big). \quad (12)$$

If the QC has l attributes and the SC has l_1 attributes and $l_1 \geq l$, then

$$QC - SC = (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l) - (A_1 = \iota_1) \wedge (A_2 = \iota_2) \wedge \cdots \wedge (A_{l_1} = \iota_{l_1}) = \bigvee_{i=0}^{l-1} \Big(\bigwedge_{j=1}^{i} (A_j = (\iota_j \cap \theta_j)) \wedge (A_{i+1} = (\theta_{i+1} - \iota_{i+1})) \bigwedge_{j=i+2}^{l} (A_j = \theta_j) \Big) \bigvee_{i=l+1}^{l_1} \Big(\bigwedge_{j=1}^{i-1} (A_j = (\iota_j \cap \theta_j)) \wedge (A_i = (\Omega_i - \iota_i)) \Big). \quad (13)$$

(Note that $\theta_j = \Omega_j$ for $j > l$ by default.)

Thus, when $l_1 \geq l$, the maximum number of terms in the disjoint canonical decomposition of the non-overlapping portion of the QC with respect to the SC is l_1. For each of these three cases the maximum number of terms in the disjoint canonical decomposition of the non-overlapping portion of the QC with respect to the SC is equal to the number of attributes in the SC.

4. Properties of access paths

Altman et al. [4] have provided the basic tools for constructing access paths through a data base and complex access-path networks can be created. Searching efficiently through such a network is a desirable aim of any information retrieval system. In order to achieve this, properties of access paths based on A-strings, E-strings, and L-strings have to be studied. In this section some important properties of access paths are discussed.

An instance of an A-string is defined over the values of the attributes. Thus the path cardinality (PC, with values PC) of an instance of an A-string is equal to the number of attributes in its exiting list. In some queries the attributes of the A-string may not all be relevant. To reflect the effect of such phenomena the frequency distribution of usage of the queries for different sets of attributes has to be considered. Analysis at that detail level is not considered in this paper. The PC of an access path, for this paper, is defined as the sum of the number of instances of A-strings, E-strings, and L-strings connected by the access path.

Suppose an E-string is defined over an A-string which is defined over the values of the attributes $A_1, A_2, \cdots, A_l$. If the multivariate frequency function of the A-string for these attributes is available, then summing the frequency function over the SC of the E-string provides the PC of the E-string. If it is assumed that the occurrences of the values of different attributes are statistically independent, then the multivariate frequency function is proportional to the product of the frequency functions of the attributes. Thus the PC of the E-string can be obtained from the frequency functions of the attributes. When the E-strings are ordered on one or more attributes, the PC of a query is calculated from the cumulative frequency functions of the attributes, which are calculated from the frequency functions.

Simple analytic forms for the PC of an E-string or an L-string can be obtained from files which have some simple distribution with respect to the attributes. One such file is the *uniform file*. In a uniform file the instances of the entity sets have uniform distribution with respect to the values of each pertinent attribute.

Suppose an A-string is defined over the attributes A_1, $A_2, \cdots, A_l$; then the number of instances of the A-string in a uniform file is given by $\prod_{i=1}^{l} |\Omega_i|$. In such a file, if n E-strings are defined on the A-string, it is not necessary to calculate the PC for each E-string to find the string with the minimum PC. Suppose $SC_1, SC_2, \cdots, SC_n$ are the string criteria for the E-strings. (In this paper, whenever there is no confusion, a string is referred to by its SC). Let $A_1, A_2, \cdots, A_m$ be the union of the SC attributes. The following function is calculated for $SC_i, i = 1, 2, \cdots, n$:

$$f_i = \prod_{j=1}^{m} C_{ij}, \tag{14}$$

where C_{ij} is the cardinality of the set of values of A_j specified by SC_i. If A_j is not specified by SC_i then C_{ij} is equal to the cardinality of Ω_j.

The left side of Eq. (14) is proportional to the PC of SC_i. The constant of proportionality is the same for all SC_i; thus, the *most favorable* SC_i, i.e., the one with the minimum PC, can be determined from the f_i. The f_i can be calculated if the C_{ij} are available. It is also possible to store the PC of each string, defined on the A-string, as parameters and then determine the most favorable SC_i. This approach was suggested by Astrahan et al. [8].

A string with an ORDER ON criterion is called an *ordered string*. In ordered strings the instances of the components are connected in an order determined by the values of one or more attributes. (In case of ordering on multiple attributes, the ordering is nested.) This property of an ordered string can be used to reduce the search length of a query if the query specifies values of the ordered attributes. In an ordered string, the search for a query can terminate when the maximum value specified by the QC for the highest ordering attribute of the SC is reached. Hence, the most favorable ordered SC for a QC is that ordered SC (among a set of ordered SC covering the QC) which ties together the minimum number of instances of the components needed to reach the maximum value specified by the QC for the ordering attribute of the SC. The most favorable SC among a set of SC depends on the query if the SC are ordered but is invariant of the QC for unordered SC.

Suppose A_i^0 is the attribute on which SC_i, $i = 1, 2, \cdots, n$, is ordered and C_{i0} is the cardinality of the set of values of A_i^0 within the SC_i. If θ_{i0} is the set of values of A_i^0 specified by a QC and if

$$f_i' = \frac{f_i}{C_{i0}} \times (\text{rank of the maximum value of } \theta_{i0} \text{ in } SC_i), \tag{15}$$

then the most favorable SC_i for the QC is the one for which f_i' is minimum.

Example 2

Suppose there are four attributes A_1, A_2, A_3, A_4. An A-string is defined over these attributes. The instances of the A-string have uniform distribution with respect to the three attributes A_1, A_2 and A_3. These three attributes can take integral values in the ranges $1 \leqq A_1 \leq 5$; $1 \leq A_2 \leq 10$; and $1 \leq A_3 \leq 20$. Thus, there are 1000 distinct instances of the A-string. It is assumed that there is no repetition of any triplet of values (A_1, A_2, A_3). Suppose two E-strings, say E_1 and E_2, are constructed with SC as follows:

E_1: $SC_1 \equiv (2 \leq A_1 \leq 5, 1 \leq A_2 \leq 8)\ 00\ A_1$;

E_2: $SC_2 \equiv (3 \leq A_2 \leq 9, 1 \leq A_3 \leq 16)\ 00\ A_2$.

Then $f_1 = 4 \cdot 8 \cdot 20 = 640$; $f_2 = 5 \cdot 8 \cdot 16 = 640$. If the two strings were not ordered, both would be equally favored.

Consider a query with $QC \equiv (A_1 = 4, 3 \leq A_2 \leq 6, 10 \leq A_3 \leq 15)$ and A_4 as the output attribute. The number of instances of the A-string in E_1 that have to be searched to answer the query (i.e., up to an including $A_1 = 4$) is $3 \cdot 8 \cdot 20 = 480$. The number of instances of the A-string in E_2 that have to be searched to answer the query (i.e., up to $A_2 = 6$) is $4 \cdot 16 \cdot 5 = 320$. Thus, E_2 is the more favorable search path for the query.

Theorem 1 If a query and a set of unordered E-strings satisfy the conditions

i. the QC and the SC are canonical terms;

ii. the QC attributes and the output attributes are a subset of the exiting list of an A-string;

iii. all the E-strings are defined over the same A-string and their SC do not negate the QC, and none of the SC are defined on the output attributes,

then

a. the search for the query can be confined to one and only one E-string if there exists at least one SC covering the QC;
b. the most favorable E-string for the query is the one with minimum path cardinality if there is more than one SC which covers the QC.

Proof Let $E(QC)$ denote the set of instances of the A-string which are relevant to the QC and $E(SC)$ denote the set of instances of the A-string which are connected by the E-string corresponding to the SC. Thus, the PC of the E-string is $|E(SC)|$. Let

$$QC \equiv (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l), \text{ and} \quad (16)$$

$$SC \equiv (A_1 = \alpha_1) \wedge (A_2 = \alpha_2) \wedge \cdots \wedge (A_{l'} = \alpha_{l'}), \quad (17)$$

where θ_i and α_i denote collections of values of the attribute A_i.

That the SC covers the QC $\Rightarrow l' \leq l$ and $\alpha_k \supseteq \theta_k$ for $k = 1, 2, \cdots, l'$; hence, from the theory of product sets it follows that $E(SC) \supseteq E(QC) \Rightarrow E(QC) - E(SC) = \phi$. Since the SC is not defined on the output attributes, the search for the QC can be confined to the E-string corresponding to the SC, which proves a of the theorem.

In an unordered string the search length for any query is equal to the PC of the string. Thus, if there is more than one SC covering the QC, the most favorable one is the one with the minimum PC. This completes the proof.

Lemma 1 If the QC of a query and a set of unordered E-strings with selection criteria SC_i, $i = 1, 2, \cdots, n$, satisfy conditions *i*, *ii*, and *iii* of theorem 1 and
iv. the SC attributes of each SC are a subset of the QC attributes but none of the SC cover the QC,
then the largest collection of instances of the A-string pertinent to the QC, in a uniform file is in the E-string for which

$$\phi_i = \prod_{j=1}^{l} \nu_{ij} \text{ is maximum,} \quad (18)$$

where $\nu_{ij} = |\theta_j \cap \alpha_{ij}|$ (if A_j is not specified in the SC_i, then $\alpha_{ij} = \Omega_j$)

$$QC \equiv (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l), \text{ and} \quad (19)$$

$$SC_i \equiv (A_1 = \alpha_{i1}) \wedge (A_2 = \alpha_{i2}) \wedge \cdots \wedge (A_{l'} = \alpha_{il'}), \quad (20)$$

where $i = 1, 2, \cdots, n$ and $l' \leq l$.

Proof The proof follows from the fact that the ϕ_i are proportional to the number of instances of the A-string pertinent to the QC and the constant of proportionality remains the same for all the SC_i. This completes the proof.

Remark When lemma 1 yields more than one E-string, the result b of theorem 1 can be applied to these E-strings to obtain a mini-max solution.

The results of lemma 1 can be generalized to the case when the QC has partial intersection with the SC and are given by the following theorem.

Theorem 2 If a query and a set of unordered E-strings, which have one instance each, with selection criteria SC_i, $i = 1, 2, \cdots, n$, satisfy the conditions *i*, *ii*, and *iii* of theorem 1, then the largest collection of instances of the A-string pertinent to the QC, in a uniform file, is the E-string for which

$$\Phi_i = \prod_{j \in L} \nu_{ij} \text{ is maximum,} \quad (21)$$

where $L \equiv$ smaller set between the set of indices of the attributes of the A-string and the set of indices of the union of the attributes belonging to the QC and the SC, $\nu_{ij}' = |\theta_j \cap \alpha_{ij}|$ [if A_j is not specified in the SC_i (and/or the QC), then α_{ij} (and/or θ_j) $= \Omega_j$] and

$$QC \equiv (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge \cdots \wedge (A_l = \theta_l), \text{ and} \quad (22)$$

$$SC_i \equiv (A_{i_1} = \alpha_{i_1}) \wedge (A_{i_2} = \alpha_{i_2}) \wedge \cdots \wedge (A_{i_{l_i}} = \alpha_{i_{l_i}}) \quad (23)$$

for $i = 1, 2, \cdots, n$.

Proof The proof is exactly similar to that of lemma 1.

The string structures defined by Altman et al. [4] do not permit splitting the instances of the component strings, but it is possible to construct access path networks in which such splitting is permissible. Access path networks which contain subsets of instances of the component strings can provide search paths with shorter PC for some queries; hence, some of their properties are begin studied. A *hierarchical nested structure of E-strings* can be constructed in the following manner:

AA: ASG $[EXL = (A_1, A_2, \cdots, A_l); ON = E_1]$;

E_1: ESG $[EXL = (AA);\ SC_1 \equiv (A_1 = \iota_1); ON = E_2]$;

E_2: ESG $[EXL = SUBD(E_1);$
$SC_2 \equiv (A_2 = \iota_2); ON = E_3]$;

E_3: ESG $[EXL = SUBD(E_2); SC_3 \equiv (A_3 = \iota_3)]$.

In this construction the instance of E_2 is obtained by subdividing the instance of E_1 [referred to as $SUBD(E_1)$] and constructing an access path through those instances of AA which satisfy SC_2. In a hierarchical nested structure of strings, the string whose EXL's do not contain $SUBD$ of a string are referred to as the strings of level 1. The strings defined over the strings of level 1 are referred to as strings of level 2, and so on. The strings which have no strings defined over them are referred to as strings at the highest level of the structure. The effect of

constructing a hierarchical nest of E-strings on the instances of the string at level 0 is the same as that of propagating the SC (of the E-strings) at different levels using AND functions. Thus, the instances of the E-string

$$E_3'\colon ESG\ [EXL = (AA);\ SC_3' = (A_1 = \iota_1 \wedge A_2 = \iota_2 \wedge A_3 = \iota_3)]$$

and those of E_3 are the same. For simplicity it is assumed that the E-strings at two successive levels do not access the same set of instances of level 0.

Suppose $\{E_i,\ i = 1, 2, \cdots, n\}$ is set of E-strings with a set of string criteria $\{SC_i,\ i = 1, 2, \cdots, n\}$ which form a hierarchical nest structure; E_1 is defined over an A-string (level 0) and E_i is defined over E_{i-1}. Let $E(SC_i)$ denote the set of instances of A-strings which are relevant to SC_i; then

$$E(SC_1) \supset E(SC_2) \supset \cdots E(SC_n).$$

Lemma 2 The search time for the instances at level 0 for a query in a hierarchical nest structure of unordered E-strings is a monotone nonincreasing function of the level of the E-string.

Proof In an unordered string the search for a query requires access to every instance connected by the string. Since $E(SC_1) \supset \cdots \supset E(SC_n)$ in a hierarchical nest structure, whatever may be the storage locations of the instances of the string at level 0, the search time for the instances at level 0 for a query using E_i cannot be greater than the search time using either E_{i-1} or E_{i-2} or $\cdots$ or E_1. This completes the proof.

Suppose for a given query with qualification criteria QC,

$$E(QC) - E(SC_1 \wedge SC_2 \wedge \cdots \wedge SC_i) = \emptyset, \text{ but} \tag{24}$$

$$E(QC) - E(SC_1 \wedge SC_2 \wedge \cdots \wedge SC_{i+1}) \neq \emptyset; \tag{25}$$

then all the instances of level 0 which are pertinent to the query can be accessed by E_i. However, all of them cannot be accessed by E_{i+1}. Thus, if the search strategy is to access all the pertinent instances of the query with one string, then E_i should be used. These results are summarized in the following theorem.

Theorem 3 The optimun (minimum access time for the instances at level 0) E-string for answering a query using a hierarchical nest structure of unordered E-strings is given by E_i where the QC of the query and the SC_i of E_i satisfy the conditions

$$E(QC) = E(SC_1 \wedge SC_2 \wedge \cdots \wedge SC_i) = \emptyset, \text{ and} \tag{26}$$

$$E(QC) - E(SC_1 \wedge SC_2 \wedge \cdots \wedge SC_{i+1}) \neq \emptyset. \tag{27}$$

Corollary 3.1 The optimum E-string for answering a query using a hierarchical nested structure of ordered E-strings is determined by the conditions of theorem 3 provided that all the E-strings are ordered on the same attribute.

Proof. If an E-string is ordered on a particular attribute which is also a QC attribute, then the search terminates when the maximum value of that attribute specified by the QC is reached. Since each E-string orders the instances of level 0 with respect to the values of the same attribute, for a fixed QC in a hierarchical nest structure, the number of instances of level 0 that have to be accessed to reach the last pertinent instance for the QC when E_i is used cannot be greater than when E_{i-1} is used. The rest of the proof follows from lemma 2 and theorem 3.

For discussing the properties of a more general string structure network, some symbols are introduced. Let $I(S)$ denote the set of instances of the string S and let $I(SC)$ denote the set of instances of the string whose string criterion is SC; $I(S/QC)$ denotes the set of instances of the string S which are relevant to the query with qualification criteria QC, and if SC denotes the string criteria, then the analogous set is denoted by $I(SC/QC)$; $I(S_1 \subset I(S_2))$ denotes the set of instances of S_1 which are contained in the instances of S_2.

Theorem 4 When the answer to a query is contained in multiple instances of a string S_1 and many strings are defined over S_1, then among them the strings desirable to the search for the query are those which satisfy one of the following conditions:

i. The string is a partitioning string for S_1 or an ordering string on all instances of S_1.

ii. The SC of the string does not affect the QC but all the instances of the string contain all the instances of S_1.

iii. The SC of the string contains the QC.

Proof Suppose a string S_2 (which is an E-string in this situation) is defined over S_1. Then the instances of S_2 are obtained by connecting together the instances of S_1. In searching, the desirable strings are those which contain the complete answers to the QC.

If S_2 is a partitioning string for S_1, then $I(S_1 \subset I(S_2)) = I(S_1)$. Thus, the set of instances of S_1 in $I(S_2/QC)$, i.e., $I(S_1 \subset I(S_2/QC))$, is equal to $I(S_1/QC)$. Hence, S_2 can be used as a search path for the QC. Similarly, when S_2 is an ordered string on all the instances of S_1, then $I(S_1 \subset I(S_2)) = I(S_1) \Rightarrow I(S_1 \subset I(S_2/QC)) = I(S_1/QC)$. Thus S_2 can be used as a search path for the QC. That the SC of S_2 does not affect a QC implies that if the instances of the components of S_2 are relevant (or not relevant) to the QC, then the instances of S_2 are also relevant (or not relevant as the case may be).

If the SC of S_2 does not affect the QC but $I(S_1 \subset I(S_2)) = I(S_1)$, then it also implies that $I(S_1 \subset I(S_2/QC)) = I(S_1/QC)$. Thus, S_2 can be used as a search path for the QC.

If the SC of S_2 contains the QC, then $I(S_1 \subset I(S_2/QC)) \supset I(S_1/QC)$. Hence, S_2 can be used as a search path for the QC.

When S_2 is defined over S_1, then $I(S_1 \subset I(S_2)) \subseteq I(S_1)$. Thus, the search for the QC using S_2 may involve access to a lesser number of instances of S_1 than when using S_1. This completes the proof.

In the above proof, if S_2 does not satisfy any of the conditions *i*, *ii*, or *iii*, then either

$$I(S_1 \subset I(S_2/QC)) \subset I(S_1/QC) \text{ or}$$

$$I(S_1 \subset I(S_2/QC)) = \phi. \qquad (28)$$

Thus, the answer to the query cannot be obtained completely from S_2. If there exists a set of strings $S_2, S_3, \cdots, S_n$ defined over S_1 such that

$$I(S_1 \subset I(S_i/QC)) \subset I(S_1/QC) \text{ for } i = 2, 3, \cdots, n$$

and

$$\bigcup_{i=2}^{n} I(S_1 \subset I(S_i/QC)) = I(S_1/QC), \qquad (29)$$

then it is possible to answer the query using the strings $S_2, S_3, \cdots, S_n$.

A string S is called the $(A_1, A_2, \cdots, A_l)$-*attribute conjugate* of strings $S_1, S_2, \cdots, A_n$ when S is defined over $S_1, S_2, \cdots, S_n$. An instance of S is obtained by tying together instances of $S_1, S_2, \cdots, S_n$ which have the same l-tuple of values for the attributes $A_1, A_2, \cdots, A_l$. If an instance of either S_1 or S_2 or $\cdots$ or S_n contains more than one distinct value of any one of the attributes $A_1, A_2, \cdots, A_l$, then the $(A_1, A_2, \cdots, A_l)$-attribute conjugate is undefined. Thus, the number of instances of S is equal to the number of distinct l-tuples of values of $A_1, A_2, \cdots, A_l$. Since the SC of S contains the attributes $A_1, A_2, \cdots, A_l$, any other string defined over S can contain only these attributes and any other attributes which have only one distinct value in each instance of S. In the rest of this paper all attributes that have a unique value in each instance of an entity set are referred to as ID (identification, with values ID) attributes[5].

Lemma 3 If a string S is a one-ID-attribute conjugate over the strings $S_1, S_2, \cdots, S_n$, then the number of instances of S is equal to the number of distinct values of the ID-attribute in the union of the instances of $S_1, S_2, \cdots, S_n$.

Proof The proof is a direct consequence of the face that each distinct value of the ID attribute in the union of the instances of $S_1, S_2, \cdots, S_n$ identifies an instance of S.

The pth accessible component of a string, defined over multiple strings, is that component of the string whose instances are accessed after those of $p-1$ components. If a query is relevant to a string S and there are multiple strings defined over it, then the most desirable string is the one in which S is the shortest accessible component, provided that none of the other strings contains additional information relevant for the query. As shown later, this condition would lead to a shorter search path.

Consider the following two L-strings:

$$S : LSG\ [EXL = (S_1, S_2, S_3);\ SC = (S_1ID = S_2ID = S_3ID);\ ON -]$$

and

$$S' : LSG\ [EXL = (S_2, S_3, S_4, S_5);\ SC = (S_2ID = S_3ID = S_4ID = S_5ID);\ ON -].$$

Here S_2 is the second accessible component in S and first accessible component in S'; hence, S_2 is the shortest accessible component in S'.

The *path cardinality for the subset of components* $S_1, S_2, \cdots, S_p$ of a string S (which is defined over a larger set of components) is defined as the number of components (of S) that have to be accessed in order to access all the elements of the subset; this is denoted by $PC(S_1, S_2, \cdots, S_p/S)$. The *path cardinality of a query in a string* S is defined as the number of instances of strings that have to be accessed when the string S is used for accessing instances (of the A-string) relevant to the query Q; it is denoted by $PC(Q/S)$. Since the complete answer to a query may not be accessible by a particular string, $PC(Q/S)$ may not be the maximum number of instances of A-strings that have to be accessed to answer Q completely.

Definition The most favorable string for a query is the one for which the path cardinality is a minimum.

Remark The path cardinality of a subset of components can easily be determined from the order of the components in the exiting list of the string.

Theorem 5 If $S_1, S_2, \cdots, S_p$ is a subset of components of a string S and $I_j(S)$ is an instance of S which is relevant to a query Q, then

$$PC(Q/I_j(S)) = \sum_{i=1}^{p} PC(Q/I_j(S_i)) + PC(S_1, S_2, \cdots, S_p/I_j(S)) + p - 1, \qquad (30)$$

where $I_j(S_i)$ denotes the instances of S_i in $I_j(S)$.

Proof The search for Q using the instance $I_j(S)$ involves accessing each component of $I_j(S)$ and checking it to see whether it is relevant to Q. If it is relevant, say to S_i,

then the access path with $I_j(S_i)$ is used to access the relevant instances of Q in $I_j(S_i)$. The path cardinality of Q in $I_j(S_i)$ is given by $PC(Q/I_j(S_i))$. When the search for Q in $I_j(S_i)$ is complete, the search control returns to S_i [i.e., where S_i is labeled as a component of $I_j(S)$] for the second time to continue the search using $I_j(S)$. Then the next component of $I_j(S)$ is checked to see if it is relevant to Q. This process continues until the last relevant component in $I_j(S)$ for Q is searched. Thus, $PC(Q/I_j(S))$ consists of

i. the sum of the path cardinality of Q in $I_j(S_1), I_j(S_2), \cdots, I_j(S_p)$;
ii. the path cardinality of the subset $S_1, S_2, \cdots, S_p$ of the components in $I_j(S)$; and
iii. the sum of the number of components in $I_j(S)$ which are accessed twice.

Therefore, we can deduce Eq. (30). This completes the proof.

Remark If the string S has multiple instances, the formula of theorem 5 has to be applied to all the instances that are relevant to the query. If these instances are not connected together somehow in the network, it may not be possible to answer the query completely.

Theorem 6 If $I'(S)$ is an instance of an E-string S which connects together instances of a string S_1, then

$$PC(Q/I'(S)) = \sum_{I_i(S_1)} PC(Q/I_i(S_1)) + PC(I_1(S_1), I_2(S_1), \cdots, I_p(S_1)/I'(S)) + p - 1, \quad (31)$$

where $I_1(S_1), I_2(S_1), \cdots, I_p(S_1)$ are the instances of S_1 which are relevant to Q and are connected together by the instance $I'(S)$.

Proof This proof is exactly similar to that of theorem 5. The E-string S is defined over the instances of the string S_1 and it is assumed that S has multiple instances. Thus, an instance of S, i.e., $I'(S)$, connects some instances of S_1 according to some string criteria. Suppose $I'(S)$ connects $I_1(S_1), I_2(S_1), \cdots, I_p(S_1), I_{p+1}(S_1), \cdots, I_n(S_1)$, although not necessarily in that order. The instances relevant to Q in $I'(S)$ are $I_1(S_1), I_2(S_1), \cdots, I_p(S_1)$.

The search in $I'(S)$ for Q consists of accessing each instance of S_1 and checking to see if it is relevant to Q. If it is relevant, say it is $I_i(S_1)$, then the access path within $I_i(S_1)$ is used to access the relevant instances of Q in $I_i(S_1)$. The path cardinality of Q in $I_i(S_1)$ is $PC(Q/I_i(S_1))$. When the search for Q in $I_i(S_1)$ is complete, the search control returns to $I_i(S_1)$ for the second time to continue the search using the access path of $I'(S)$. Then the next component of $I'(S)$ is checked to see if it is relevant to Q, and so on. This process is continued until the last relevant instance for Q has been searched. By using the same summing technique as in the proof of theorem 5 it follows that

$$PC(Q/I'(S)) = \sum_{I_i(S_1)} PC(Q/I_i(S_1)) + PC(I_1(S_1), I_2(S_1), \cdots, I_p(S_1)/I'(S)) + p - 1. \quad (32)$$

This completes the proof.

In many situations the catalogue may not provide a clue for calculating $PC(Q/I_i(S_1))$ or $PC(I_1(S_1), I_2(S_1), \cdots, I_p(S_1)/I'(S))$ or even for determining the set $I_1(S_1), I_2(S_1), \cdots, I_p(S_1)$. Thus, additional information has to be stored so that these parameters can be determined. Usually the QC of the query and the SC of the string are specified in terms of attributes and their values. Thus, if the string is ordered on the appropriate attribute and the ranks of the values specified by Q for that attribute in S_1 are known, then these parameters can be determined. If the instances of S_1 are not ordered on a QC attribute or if the rank of the maximum value of the ordered QC attribute cannot be determined in S_1, then $PC(I_1(S_1), I_2(S_1), \cdots, I_p(S_1)/I'(S))$ cannot be determined and its value may have to be chosen to be equal to the number of instances connected by $I'(S)$. If it is not possible to determine whether a particular instance, say $I_i(S_1)$, is relevant to Q by examining it (i.e., its encoded representation), then further search of the access path defined by $I_i(S_1)$ has to be performed to determine whether $I_i(S_1)$ is relevant to Q. In such cases the term $p-1$ in theorem 6 has to be replaced by $n-1$ where n is the number of instances of S_1 connected together by $I(S)$.

The path cardinality of access paths in string networks that have some specific characteristics can be calculated by more simple techniques. One such string network is a *tree-string structure*. A tree-string structure is a tree-structure of access paths that are constructed by using a series of E-strings or L-strings or a combination of both. Here we discuss only a tree-string structure constructed by using E-strings alone. An example of such a tree-string structure is as follows

Example 3
Suppose $\{I_i(S)\}$ denotes the set of instances of a string S where each instance is parametrized by the values of the attributes $A_1, A_2, \cdots, A_l$. The instances of any E-string defined over S can be parametrized by the values of $A_1, A_2, \cdots, A_p$, where $p < l$. Suppose E_1 is defined over S with a parametrized string criteria

$$SC(v_1, v_2, \cdots, v_p) \equiv (A_1 = v_1) \wedge (A_2 = v_2) \wedge \cdots \wedge (A_p = v_p); \quad (33)$$

i.e., each distinct p-tuple $(\nu_1, \nu_2, \cdots, \nu_p)$ corresponds to an instance of E_1. Similarly, E_2 is defined over E_1 with parametrized string criteria

$$SC(\nu_1, \nu_2, \cdots, \nu_{p-1}) \equiv (A_1 = \nu_1) \wedge (A_2 = \nu_2) \wedge \cdots \wedge (A_{p-1} = \nu_{p-1}); \quad (34)$$

E_p is defined over E_{p-1} with $SC(\nu_1) \equiv (A_1 = \nu_1)$; E_{p+1} is defined over E_p. Here E_{p+1} is the root of the tree-string structure and the instances of E_1 are the leaves of the tree.

Multilevel sorting can be implemented by an *ordered* tree-string structure. The ORDER ON specification in the SC is used to construct the ordered tree-string structure as follows: The string criterion of E_{p-i+1} is given by

$$SC(\nu_1, \nu_2, \cdots, \nu_i) \equiv (A_1 = \nu_1) \wedge (A_2 = \nu_2) \wedge \cdots \wedge (A_i = \nu_i, 00\ A_{i+1}). \quad (35)$$

The SC of E_{p+1} is $(00\ A_1)$ and the SC of E_1 may or may not have an order on specification.

Suppose there are n instances of E_p and the number of instances of E_{p-k-1} within the j_kth instance of E_{p-k} which is within the j_{k-1}th instance of E_{p-k+1} and so on $\cdots$ which is within j_1th instance of E_p is denoted by $n_{j_1 j_2 \cdots j_k}$. Then the path cardinality of the longest search path in an ordered tree-string structure is given by

$$PC_{\max}(T_S) = 2n - 1 + \sum_{j_1} (2n_{j_1} - 1) + \sum_{j_1} \sum_{j_2} (2n_{j_1 j_2} - 1) + \cdots + \sum_{j_1} \sum_{j_2} \cdots \sum_{j_{p-2}} (2n_{j_1 j_2 \cdots j_{p-2}} - 1) + \sum_{j_1} \sum_{j_2} \cdots \sum_{j_{p-1}} n_{j_1 j_2 \cdots j_{p-1}}. \quad (36)$$

If PC of the search for the query with $QC \equiv (A_{i_1} = \theta_1) \wedge (A_{i_2} = \theta_2) \wedge \cdots \wedge (A_{i_k} = \theta_k)$ has to be determined, then the $n_{j_1 j_2 \cdots j_m}$ have to be replaced by the ranks of the maxima of the values of the attributes specified by the QC within the appropriate instances of the E-strings.

5. Algorithm for search path

In any efficient network every node cannot be used as entry points to the network. Hence, each network has associated with it a search path algorithm. The search algorithm proposed here for a query in the access path network attempts to obtain a search path with minimum path cardinality. The shortest path is not attempted because the algorithm would need too much storage. The algorithm is based on the assumptions that the QC are given in the canonical form and the SC of the E-strings and L-strings are given as the unions of canonical terms. The algorithm outlines procedures for handling only one canonical term in the SC. If there are multiple canonical terms in the SC, then the matching of the QC has to be performed with each term of the SC. If one of the terms matches or covers the QC, then the SC matches or covers the QC. If all the terms negate the QC, then the SC negates the QC. The algorithm is composed of three parts which are denoted A1, A2 and A3.

- *A1: Query on one entity set*

Step 1

Locate the A-strings which are defined over all the QC attributes and the output attributes. If there are no such A-strings, then search for multiple A-strings with ID conjugates (or ID conjugates over E-strings defined over the A-strings with SC, non-negating the QC. As the query is based on one entity hence these relevant E-strings cannot connect more than one instance of an A-string) with this property. If no such A-string is available, then search for multiple A-strings with the same ID attribute which covers all the QC attributes and the output attributes. If no such A-string is available, then the query cannot be answered by the access path network.

Step 2

If there exist one or more A-strings defined over all the QC attributes and the output attributes, then proceed to the following substeps.

Substep 2.1 Examine the E-strings defined over each of these A-strings. If there exists only one E-string whose SC covers the query, then select it. If there is more than one such E-string, then select the one which has the most favorable SC covering the query (results of theorems 1 and 2 may be needed). Then search for the entry point of the E-string by using algorithm A2.

Substep 2.2 If there is no E-string covering the query satisfying substep 2.1, check to see if there exists a set of E-strings defined over the A-strings whose SC do not negate the QC. Then obtain a canonical disjoint decomposition of the QC with respect to these E-strings (i.e., from $QC - SC$). If the set of E-strings provides a complete decomposition of the QC and covers the output attributes, then a cover for the query has been obtained. If the set of E-strings provides a partial decomposition of the QC and/or a partial cover for the output attributes, then the residual portion over the A-strings. In such cases the substep 2.3 also has to be executed, otherwise the search for the entry point of the E-string is performed by using algorithm A2.

Substep 2.3 If there is no E-string defined over the A-strings or a cover for the query has not been obtained from substeps 2.1 or 2.2, then the L-strings defined on the A-strings are examined. Select only those L-strings which form ID conjugates. The procedures described

for the A-strings in substeps 2.1 and 2.2 are then applied to each of the L-strings with respect to the query or the residual QC and or the residual output values (if substep 2.2 has not resulted in a cover for the query). If there is more than one E-string defined on the L-string which covers the query (or the residual QC and/or the residual output values, as the case may be), then the one with the minimum PC is selected. If there is no E-string defined on the L-string, this substep (i.e., 2.3) has to be repeated until an E-string is obtained. Then the search for the entry points of the E-strings are performed by algorithm A2.

Step 3

If there exist multiple A-strings with ID conjugates (i.e., L-strings) such that there are multiple L-strings defined over them and each of these covers all the QC-attributes and the output attributes, then the procedures described for A-strings in step 2 and its substeps are applied to each L-string. If there is more than one qualified L-string, the one with the minimum PC is selected. (Results of theorems 5 and 6 may be needed.) Then the search for the entry points for the relevant strings is performed by algorithm A2.

Step 4

If a sequence of ID conjugate L-strings over multiple A-strings results in an L-string which covers all the QC-attributes and the output attributes, then the procedures described for A-strings in step 2 and its substeps are applied to this L-string. If E-strings are mixed with L-strings to create an L-string which covers all the QC-attributes and the output attributes, the SC of the E-strings are checked to determine that they do not negate the QC. Then step 2 and its substeps are applied.

Step 5

If there exist multiple A-strings with the same ID attributes but not ID conjunction between them, then apply step 2 to each A-string with respect to the segment of the QC which is relevant to that A-string. Then search for the entry points using algorithm A2. When the sets of instances of the different A-strings relevant to the query are retrieved, an ID-conjugation has to performed by matching to obtain the data relevant to the query.

Step 6

If steps 2 through 5 are not applicable to a query with respect to the access path network, then the query cannot be completely answered from the network.

••*A2: Entry point search*

Results of theorems 4, 5, 6, and 7 have been used to derive this algorithm.

Case 1

For one E-string or L-string which has one instance only: Use the *ON* criterion to locate the string or strings (there have to be L-strings) defined on it. Repeat this process for all the new L-strings. Continue the process until one or more entry points are reached. As the process is continued a record of the different access paths and their PC (using theorem 5 and 6) is kept. The best choice is the path with the minimum PC.

Case 2

For multiple E-strings or L-strings (all of them need not cover the A-strings) which have one instance each: The method is explained for multiple E-strings with one instance each; the same method is applicable to L-strings or combinations of L-strings and E-strings. Suppose the E-strings are $E_1, E_2, \cdots, E_n$. Choose any one, say E_1, as the starting point. Then examine the L-strings defined on E_1. Examine the components (from the exiting list) of each L-string and include those components which contain any member of the set $E_2, E_3, \cdots, E_n$ in the relevant access paths. If some of the components are L-strings, then the same process is repeated for those L-strings. In constructing relevant access paths, only those strings which are defined over nonredundant relevant E-strings are included. Thus, a set of L-strings defined over E_1 which have access paths to a subset of $E_1, E_2, \cdots, E_n$ can be selected. The relevant access paths associated with each L-string are recorded and their PC are calculated by using theorems 5 and 6. The same process is repeated with the new L-strings until entry points are reached. This process associates with each entry point a set of favorable access paths to $E_1, E_2, \cdots, E_n$ or its subset. Then a set of paths covering $E_1, E_2, \cdots, E_n$ is selected on the basis of minimum PC.

Case 3

For one E-string or L-string which has multiple instances: If there is a set of E-strings or L-strings with one instance each defined over the given string which forms a cover for the relevant instances, then the method of case 2 is applied to these new strings. If there is a set of E-strings or L-strings, each with multiple instances, defined over the given string and the set forms a cover for the relevant instances, then the method of case 4 has to be applied. If there is only one ID-conjugate L-string defined over the given string, then the method for this case (i.e., case 3) has to be applied again to the new string. If there is one E-string or L-string with one instance defined over the given string which forms a cover for the relevant instances, then the method of case 1 is applied to the new string. If all the strings defined over the given string, taken together, do not form a cover for all the desired instances, then the search cannot be completed.

Case 4

For multiple E-strings or L-strings which have multiple instances: Use the *ON* criterion of the given strings and the SC of the strings defined on it to find a set of strings which covers all the desired instances. Then apply the methods for cases 1, 2, and 3 to the new strings to find the best access paths for the desired instances. In some situations this method may have to be applied repeatedly to subsets of the set of desired instances. If there does not exit a set of strings which covers all the desired instances, then the query cannot be answered completely.

• *A3: Query on multiple entity sets*

This algorithm is based on results of algorithms A1 and A2.

Step 1

Locate all the A-strings defined over subsets of the QC attributes and output attributes.

Step 2

Use algorithm A1 on the A-strings associated with each entity set with respect to the segment of the QC which is relevant to that entity set. During this process, if L-strings are encountered which connect other relevant entity sets, then modify the segment of the QC accordingly and check for its non-negation. If there is more than one access path over the same subset of the relevant entity sets, then select the access path with the minimum PC.

Step 3

Apply algorithm A2 to the output strings obtained from step 2. Among the access paths obtained from A2 only those which access strings on all the desired entity sets (i.e., somewhere along the path there exist appropriate L-string conjunctions between strings defined over all the relevant entity sets) are relevant paths. Among these the one with the minimum PC is selected.

Step 4

If no such access path is obtained from step 3, then an attempt should be made to access the desired entities by different access paths and then to obtain the relevant data for the query by matching.

Example 4

To illustrate the algorithm consider the data set and the string network outlined in example 1. Suppose the query is of the form: Retrieve the average salary of the person with man number 74672. Thus, the QC attribute is *Man* and the output attribute is *Av Sal.* Now we apply algorithm A1. Using step 1 we find that there are two A-strings, namely $JA1$ and $JA3$, which are relevant to the query.

Then we apply step 2 of A1. This leads to substep 2.1. The E-string $JE1$ is defined over $JA1$ and it covers the query. The E-string $JE2(n)$ is defined over $JA3$ and it also covers the query. Both $JE1$ and $JE2(n)$ connect the same A-string instances although they do not have the same PC.

Now apply algorithm A2. Case 1 of A2 is applied to $JE1$. There is no string defined over $JE1$; hence, if $JE1$ is an entry point, then $JE1 \rightarrow JA1$ is an acceptable search path. If $JE1$ is not an entry point, then this path cannot be used for retrieval. Case 3 of A2 is applicable to $JE2(n)$. There is one L-string, viz. $JL1$, defined over the relevant instances of $JE2(n)$; hence, case 3 has to be applied again to $JL1$. There is one E-string with one instance, viz. $JE3$, defined over $JL1$; hence, case 1 is to be applied to $JE3$. If $JE3$ is an entry point, another alternate search path is obtained. Then the PC of the two paths are calculated by using theorems 5 and 6 to determine the best search path for this query.

6. Conclusion

Logical relations and query sets can induce a large number of strings on a given set of data, which can result in a very complex access path network. Thus, analyzing all properties of string structures and providing the best search path algorithm for all situations are extremely difficult. When placement rules for data on complex storage media and migration of data are taken into consideration, the problem becomes still more complex. In this paper we have provided a solution to the search path problem for any general network constructed using A-strings, E-strings and L-strings, and the algorithm is based on minimum path cardinality. The algorithm is suitable for core-type storage media and is quite complex because the network is of very general nature. Simple algorithms can be deduced from this algorithm if the network has some systematic structure. However, the problem of constructing the optimum network for a given set of likely queries and the effect of placement algorithms on search paths are still open questions.

7. Acknowledgments

The authors thank E. B. Altman, M. M. Astrahan, P. L. Fehder, and A. D. Inselberg for valuable discussions during this work.

Appendix

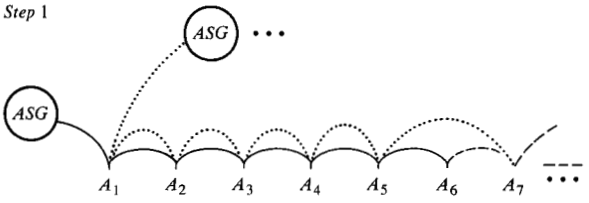
Pictorial diagrams of the types of string networks covered by the steps of the algorithms are given below. For this purpose

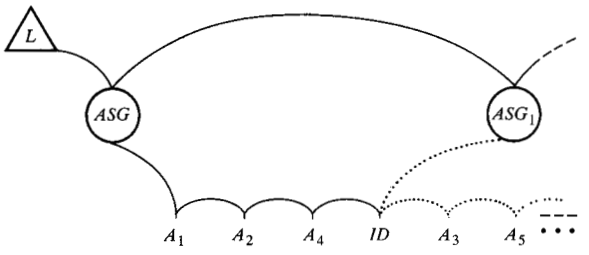
$$QC \equiv (A_1 = \theta_1) \wedge (A_2 = \theta_2) \wedge (A_3 = \theta_3)$$

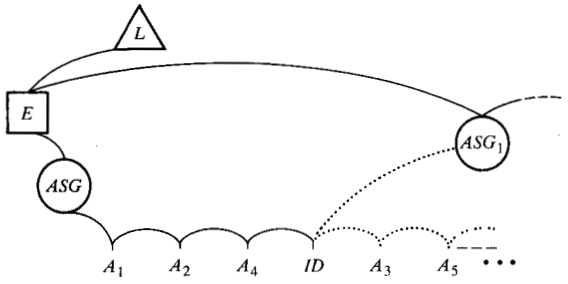
and the output attributes are A_4 and A_5.

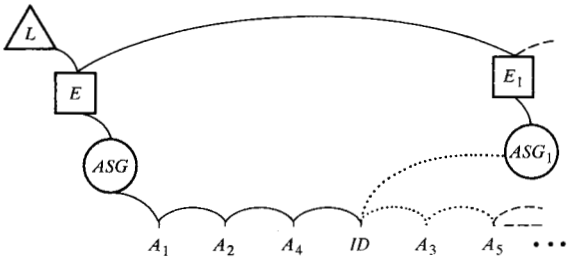
Algorithm A1

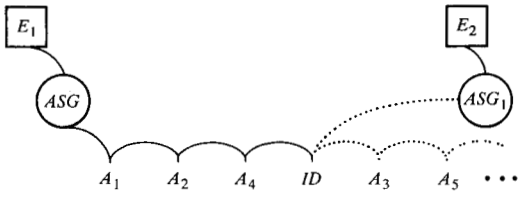
Step 1



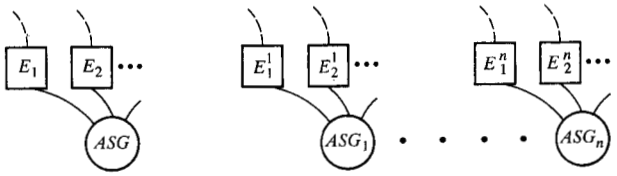






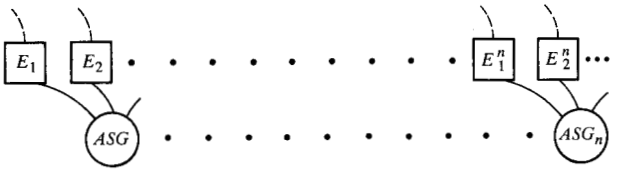


Substep 2.1



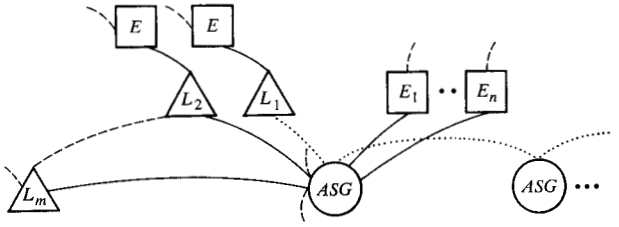
and one *SC* covering the query.

Substep 2.2

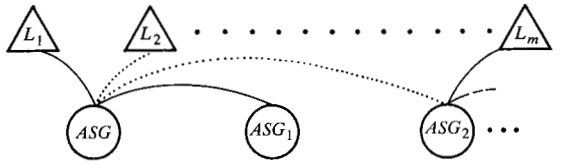


and decomposition of the *QC* w.r.t. the *SC*'s.

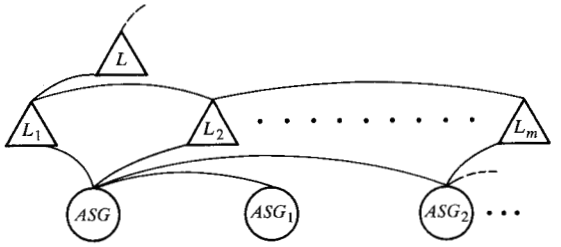
Substep 2.3

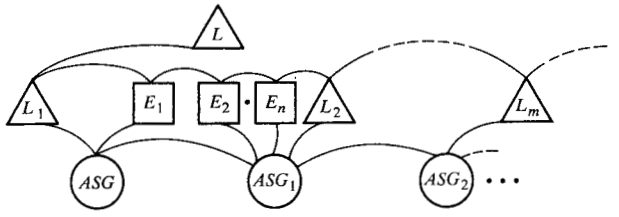


Step 3

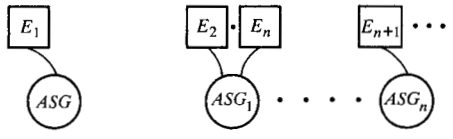


Step 4



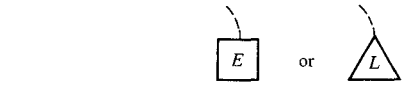


Step 5

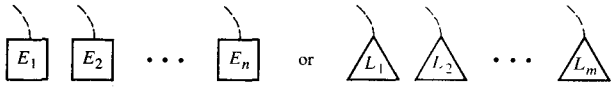


Algorithm A2

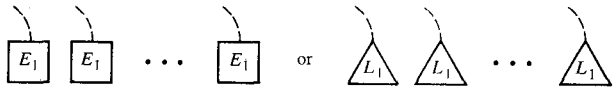
Case 1



Case 2

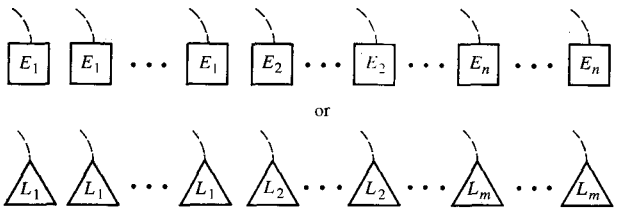


Case 3



or any combination.

Case 4



or any combination of these.

References

1. CODASYL Data Base Task Group, Report to the CODASYL Programming Language Committee, *Report CR11*, 5(70) 19, 080, Association for Computing Machinery, New York, 1969.
2. M. E. Senko, E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structures and Accessing in Data-Base Systems," *IBM Syst. J.* **12,** 30 (1973).
3. P. L. Fehder, "The Representation Independent Language, Part 1: Introduction and the Subsetting Operation," *Research Report RJ 1121*, IBM Research Laboratory, San Jose, California, 1972.
4. E. B. Altman, M. M. Astrahan, P. L. Fehder, and M. E. Senko, "Specifications in a Data Independent Architectural Mode," *Proceedings of the ACM SIGFIDET Conference*, Denver, Colorado, 1972.
5. D. E. Knuth, *The Art of Computer Programming, Fundamental Algorithms*, Vol. 1, 1969, and *Sorting and Searching*, Vol. 3, 1973, Addison-Wesley Publishing Co. Inc., Reading, Massachusetts.
6. S. P. Ghosh and M. M. Astrahan, "A Translator Optimizer for Obtaining Answers to Entity Set Queries from an Arbitrary Access Path Network," *Proceedings of IFIP Congress*, Stockholm, Sweden, August 1974, to be distributed by North-Holland Publishing Company, Amsterdam, Netherlands.
7. M. M. Astrahan and S. P. Ghosh, "A Search Path Selection Algorithm for Data Independence Access Model (DIAM)," *Proceedings of the ACM SIGFIDET Workshop*, Ann Arbor, Michigan, 1974, to be published.
8. M. M. Astrahan, E. B. Altman, P. L. Fehder, and M. E. Senko, "Concepts of a Data Independent Architectural Model," *Proceedings of the ACM SIGFIDET Conference*, Denver, Colorado, 1972.

Received December 6, 1973; revised April 10, 1974

S. P. Ghosh is located at the IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California 95193; M. E. Senko is at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.