NetBook - a data model to support knowledge exploration

Dennis Shasha

Courant Institute, New York University
251 Mercer Street, NY, NY, 10012, USA.
(shasha@nyu-csd2.Arpa)

Abstract

Knowledge exploration is the activity of finding out what other people have thought about. Normally, people explore knowledge by reading books or articles or by talking to other people. This paper discusses an alternative approach: a system whose knowledge is in the form of text fragments plus a query language to help users access appropriate fragments. Drawing primary inspiration from database theory, hypertext systems, knowledge representation, and a study of textual fragments called fragment theory, the paper describes and motivates a data model to support knowledge exploration.

1. Books vs. People

What is the difference between hiring a consultant and reading his or her book? You can follow your own line of inquiry in asking the consultant questions. The consultant will answer you without telling you much irrelevant or redundant material. The book may also answer the questions, but to find the answers you have to be skilled and lucky in using the index or you have to read a good part of the book.

The objective of the NetBook project is to replace books and documents by a collection of text fragments,[1] relations among fragments, and a query language. An explorer uses the query language to retrieve a text fragment or fragments.

The NetBook project does not attempt to break new ground in natural language understanding by computer. Therefore, the reader's queries are in an artificial language, and the system's responses consist of text fragments that writers have entered as is. The challenge is to design a concise data model that incorporates necessary facilities for such a system.

1.1. Motivation and Related Work

With numbing frequency, we hear about the ongoing and worsening "knowledge explosion." With so much knowledge around, a system to support knowledge exploration should consider the explorer's time and effort to be the primary measure of cost.

Systems that address the knowledge exploration problem come from the fields of information retrieval, database management, natural language understanding, computer-aided instruction, and non-linear text systems known as hypertext.

Information retrieval systems help users find documents relevant to their interests. Increasingly, they depend on keyword search (possibly with automatic synonym generation) in addition to standard library classification techniques. The main attraction of such systems is that they do not require special effort by the reader or writer. Specially trained professionals (usually librarians) do the classification step and keyword retrieval systems are widely available. However, studies have shown that different indexers use different classifications [ZD69]. This has led some studies to argue that indexing may be worse than keyword retrieval [S70]. However, a recent experiment [BM85] shows that a state-of-the-art keyword retrieval system retrieves less than 20 percent of the documents relevant to a particular search (mostly because people may refer to the same thing in different ways; one person's "accident" is another person's "difficulty"). One must conclude that neither manual indexing done by professional classifiers nor keyword searching nor even their combination are altogether satisfactory for retrieving relevant information.

Database management systems also help explorers retrieve information. They provide an important paradigm for knowledge exploration systems. However, there are several important differences between the two applications:
1) Database users can remember the elements of a database management system schema that they are likely to run across. Users of a knowledge exploration system by contrast may have no prior knowledge of the schema. Moreover, the schema may be huge.
2) Database entities (e.g. tuples) are quite small (say a few hundred characters); in contrast, entities in a knowledge exploration system which may be several paragraphs long and may have attributes of that length as well.
3) Database tuples can be understood without referring to other tuples; by contrast, a knowledge system must take into account a prerequisite relationship between some text units and others.

The goal of natural language understanding systems is to build up automatically an internal representation of text in

[1] What we call 'text' in this document may in fact be a picture, an experimental simulation, or a live performance. We use the term text for the sake of concreteness.

order to answer questions as a human would. Unfortunately, the research is far from achieving this goal even for highly restricted domains [GHF82, S81, SA77, WH81]. Our project does not assume even a partial solution to this problem, though we do point to some areas where advances in natural language understanding would be helpful.

In contrast to natural language understanding and information retrieval, intelligent computer aided instruction requires the knowledge producer to tailor the knowledge to the explorer (student) rather than merely to write an article and depend on a computer and other people to act as intermediaries. The best of these projects [BRB76, C76, SCG78] allow a student to test himself or herself on simulators and comment on how reasonable the student's hypotheses are. Our project adopts the philosophy of tailoring information to the explorer, but the tailoring requires less expertise and effort.

Partly in reaction to the difficulty of natural language understanding and partly because it seems to be an attractive and liberating approach to exploring knowledge, non-linear text forms have caught the imagination of researchers [B45, B82, E84, EWN73, MM79, N79, Nel83, T83, Wey82]. These so-called *hypertext* systems consist of text fragments embedded in a directed graph with labelled edges and instructions allowing the user to traverse edges. They also allow users to make commentaries on what they read [L85, T83]. Hypertext systems often incorporate state of the art technology in their user interfaces, such as high resolution graphics, sophisticated pointing devices, and even pictorial simulations. However, without a content-based query system, hypertext users must wander through text units trying to find relevant information; once they find it, they must figure out an order in which to read it; and after they have finished reading, they have no idea whether there is more. This may explain why such systems has few users despite the idea's long history [B45].

Some database systems [C80, H80, SK82, F84] have adopted a hypertext approach in the sense of allowing users to wander among the information in a database, but it is too early to gauge the direction of this approach.

2. Fragment Theory for Retrieval

The system we envision consists of text fragments containing one or more *information units*. Each information unit is an idea or a fact. Different people may disagree about what are information units, but the qualitative conclusions of the model are independent of those differences.

2.1. Linear text contains redundancy or irrelevance

Consider the following text fragments:

1) When the red alarm rings, look around for fire. Also, check for smoke, strange smells ....

2) When the blue alarm rings, see if there are any thieves. Also check for forced locks, open doors ....

3) Leave the building. Take out the following documents: ....

4) [If the red alarm rang,] use the following route to the fire station....

5) [If the blue alarm rang,] use the following route to the police station.....

This might be represented in linear text in basically two ways (1, 2, 3, 4, 5) or {(1, 3, 4),(2, 3, 5)} -- where parentheses imply sequence and curly brackets imply that either order is possible. A reader of this information might be interested in the red alarm, the blue alarm, or both. Readers interested in exactly one of the two alarms would be well satisfied with the second representation, particularly if the conditionals in 4 and 5 were removed. However readers interested in both alarms would be displeased by the repetition of 3. Similarly, readers interested in both alarms would be happy with the first representation, but readers interested in one alarm would be displeased by the irrelevant information about the other alarm.

text: (1, 2, 3, 4, 5); choice: (2, 3, 5); irrelevant: {1, 4}

text:((1, 3, 4),(2, 3, 4)); choice:(1, 2, 3, 4, 5); redundant:{3}

Figure 1: There are choices for which most linear texts will have redundancies or irrelevances.

The main lesson is that when different readers are interested in distinct but overlapping pieces of information, some of them will encounter irrelevance or redundancy when reading linear text, unless there is a representation tailored to every interest. This motivates our search for a system that would permit us to store text in fragments, then present them appropriately. For example, if the reader asks about the red alarm, present (1, 3, 4), if the reader asks about both, present (1, 2, 3, 4, 5).

2.2. Measuring Redundancy and Irrelevance

Even when the text is stored as fragments which are optimally chosen to answer explorer's questions, explorers may encounter redundancy or irrelevance. For example, the route to the fire station and to the police station in items 4 and 5 may begin with common instructions, so the presentation (1, 2, 3, 4, 5) would contain some redundancy even for the user who wants to know about both alarms. Let us call the common route information c and the remaining information in 4 and 5 is 4' and 5' respectively. Thus, presenting (1, 2, 3, 4, 5) really means presenting (1, 2, 3, (c, 4'), (c, 5')). The second c is redundant. Here, {1, 2, 3, c, 4', 5'} are fundamental information units; {1}, {2}, {3}, {c, 4'}, and {c, 5'} are groupings of information. The choices are subsets of the information units: {1, 3, c, 4'}, {2, 3, c, 5'}, {1, 2, 3, c, 4', 5'}, representing units relevant to the red alarm, the blue alarm, and both alarms.

In general, the redundancy and irrelevance of a set of groupings Q that cover a choice S[2] are defined as follows:[3]

$$Redundancy_{Q,S} = \sum_{q \in Q} |q \cap S| - |S| \text{ and}$$

$$Irrelevance_{Q,S} = \sum_{q \in Q} |q - S|.$$

2.3. Grouping selection problem

Even in this idealized model the problem of choosing groups to minimize the sum of the redundancy and irrelevance (which together constitute unwanted material) is hard. This problem might arise when a teacher wants to assemble groups

[2] - Covering means that every element of S is a member of at least one of the sets in Q.

[3] - Common software engineering practices such as information hiding and context-dependent help facilities reduce irrelevance as we define it here. Relational selection and projection reduce irrelevance by removing unwanted attributes and tuples. Projection reduces redundancy by eliminating duplicates.

for presentation to a student. We frame this as a decision problem.

Grouping Selection problem.
Instance: Let I be a finite set (of "information" units). Let *Grouping* be a collection of subsets of I, (the "groupings" of information units). Let S be a subset of I, the "choice units." Let *unwanted* be a positive integer.
Question: Is there a subset Q of *Grouping* such that Q covers S and $redundant_{Q,S} + irrelevant_{Q,S} \leq unwanted$?

Verifying a proposed solution is at worst quadratic in the number of information units in the members of Q, so the problem is clearly NP-easy [GJ79].

Theorem Group Selection: The grouping selection problem is NP-hard [S85].

In spite of this negative result, many heuristics can be tried. For example, if the sets in Grouping are all disjoint, there is an optimal solution to the problem -- just pick the sets in Grouping that intersect the choice set S. More generally, if any element in S is in only one group, then pick that group. After that, pick groups in descending order based on the ratio of *new/extra*, where *new* is the number of new information units found in a group and *extra* is the increase in redundancy and irrelevance resulting from choosing that group.

Suppose we have choices { {2,4}, {3,4,5}} and groupings A = {1,2,3}, B = {2,4,5}, C = {1,4}, D = {3,5} , and E = {2,5}. The heuristics suggest that for {2,4} we choose C and E. For {3,4,5} we choose D and C.

2.4. Predicate Selection Problem

The grouping selection problem arises when a person wants to assemble a set of information groups for someone else. However, an individual exploring on his own for a choice c will use a query language offering a collection of predicates (corresponding to selection, projection, and so on in relational algebra) from which the user can choose some boolean combination.

The explorer should choose a boolean combination P such that any information unit satisfying c will satisfy P. Thus, the predicate selection problem is to select a predicate covering a choice that minimizes redundancy and irrelevance. Here, redundancy and irrelevance are measured with respect to the set of groupings that satisfy P. The problem assumes no knowledge of the entire set of groupings G, but only of the predicates.

For example, suppose we want to know all the synchronization mechanisms used in concurrent programming languages. In the absence of special information about these languages, we would have to look at the synchronization mechanism used by every concurrent programming language. Thus, the predicate would be "synchronization mechanism of a concurrent programming language." The grouping selection problem assumes information about the set of groupings G, so an optimal set of groupings for this choice may include the synchronization mechanisms for only a subset of the languages. So, predicate selection will in general yield more redundancy and irrelevance than grouping selection, because less information is available to the explorer. (The formal similarity of the two problems, however, implies that predicate selection is NP-complete.)

We model the construction of a query as a predicate selection. The question is: how can we design the information groupings so commonly selected predicates will encounter a "reasonable" amount of redundancy or irrelevance?

3. Applications of fragment theory

Breaking up groups into component information units reduces irrelevance and redundancy but is difficult. Our criterion for choosing the groupings to break up (decompose) is based on the ratio of unwanted to useful information generated by common predicates.

3.1. Decomposition

For example, suppose that each concurrent programming language has properties design history and synchronization mechanism. Suppose that questions about both are frequently asked. The question is which of the two properties could be kept with information specific to each language?

There is a one-to-one relationship between languages and design histories, whereas there is a many-to-one relationship between languages and synchronization mechanisms. Suppose there are m languages, m design histories, and n synchronization techniques used by the languages $n<m$. Then there are $O(m)$ information units about the design histories of the languages, but only $O(n)$ about synchronization mechanisms. Keeping either of these with information specific to each language results in $O(m)$ irrelevance and less redundancy. Hence the ratio of unwanted information to desired information is $O(1)$ for design history and $O(m/n)$ for synchronization mechanism. This suggests separating the synchronization mechanism from the language description.

A misinterpretation of this example is that redundancy in the information stored is always bad. In fact, redundancy is bad only if the groups with the redundant information are likely to be selected by the same predicate. For example, a history of Germany and a biography of Bismark may both contain sections on the role of Bismark in the unification of Germany, but no single predicate will likely select them both.

The programming language example suggests that groupings should be decomposed so that often-chosen predicates that return a multiset of information units, return instead a set without duplicates. The Bismark example illustrates that achieving this decomposition for predicates p and p' doesn't guarantee that it will hold for $p \vee p'$, because distinct information groupings may have the same information units. However, the redundancy grows slowly, as the following observation states.

Observation about decomposition: Suppose every information unit contained in any grouping satisfying predicate p is in only one such grouping and similarly for p'. Then the same property holds for $p \wedge p'$ and there are at most two such groupings containing the same information unit in $p \vee p'$. Moreover the irrelevance in $p \wedge p'$ is bounded above by the minimum of the irrelevance in p or p'. The irrelevance in $p \vee p'$ is bounded above by the sum of irrelevances of p and p'.[]

3.2. Soft Aggregates

A major difficulty facing the knowledge explorer is to filter out information that is relevant, but more detailed than necessary. Database management systems use aggregate functions to handle this problem. Knowledge exploration entails an additional mechanism which we illustrate below.

Suppose we want to know about some geographical region, say the high Oregon desert, and the system has detailed information about the municipalities in the area. Reading about all the municipalities would be dull and redundant. Therefore, we would want to use aggregates. Some of

these are standard quantitative aggregates such as the average rainfall of the municipalities. Others entail a minor extensions of these aggregates such as the union of floral species in the area. However others are "soft" aggregates, e.g. the region has little rainfall, but a good deal of water flowing from the mountains to the west. We might come to this generalization by reading about each municipality, but the ratio of unwanted to useful information would be too high. Our data model handles this situation by positing a partial order[4] "subvalue" on the entities of each type. For example, the municipalities of the high Oregon desert would be subvalues of counties there. This allows explorers to discover qualitative generalizations without having to wade through much useless information.

4. Data Model: Retrieval

Our data model follows the spirit of the functional data model Daplex [Ship81, CDFR83] and some aspects of the semantic data models of [LM79]. Other features seem specific to the problem.

The basic information groupings are called *coherent units*. A coherent unit is understandable to a reader who understands the terms it uses. Thus, a coherent unit may contain proper nouns and technical terms, but no anaphoric references (such as pronouns) to other text. We make this requirement so that a particularly knowledgeable reader may start at any coherent unit.[5] Each coherent unit is organized as a tree of text fragments in which any path from the root text fragment to a descendant text fragment is coherent, but the non-root nodes need not be coherent on their own.

The basic constructs of the query language are *entities*, *types*, and *functions*. Entities are normally entire coherent units, though they may consist of the text fragments along the path from the root of a coherent unit to an interior node. The functions take entities as their domain and returns a single value or a set of values. Such values may be entity identifiers or may be drawn from scalar, vector, or character data types. Thus, functions correspond to properties of entities (in general, a function may have many entities as arguments but we ignore this case for the sake of explanation). A type is a collection of entities that all belong to the domains of the same functions.

The set of functions defined on a type induces a partial order, called *subtype*, defined as follows: if the set of functions defined on type t is a subset of the set defined on t', then t' is a subtype of t. Intuitively, entities of type t' have more properties than entities of type t.

The major structural difference between our model and Daplex is the notion of a partial order *subvalue* on values. Value a is a subvalue of b with respect to a type t if any entity with value a of t also has value b. For example, if asking about shipments to Sweden we would be interested in shipments to Stockholm, a subvalue of Sweden in a geographical region type.

4.1. Retrieval Operations

In this model, types and functions as well as entities may be targets of a query. These operations allow the explorer to discover the connections to an entity as in semantic data models [LM79].

A query is a calculus expression [U82] of the form {target : qualification}.[6] The target consists of two tokens. The first is either **type** or **entity** or **function**. The second is a variable, called the *target variable*. The qualification is a formula whose only free term is the target variable. Variables may represent types, entities, or functions.

The *atoms* of a formula are:

1. { entity variable | entity constant} {is a subvalue of | = | $\in$ | θ } { entity variable | entity constant}, where one of the entity arguments must not be a constant. The symbol θ represents an arithmetic comparison operator (possible only if the two entity arguments are comparable scalar values).

2. { entity variable | entity constant} is of type {type variable | type constant}, where one of these must not be a constant.

3. {type variable | type constant} {is a subtype of | = } {type variable | type constant}, where one of these must not be a constant.

4. {entity variable} {has keyword} {keyword expression, some regular string expression}.

Formulas are made up of atoms as follows:

1. Every atom is a formula.

2. If $form_1$ and $form_2$ are formulas then $form_1 \wedge form_2$, $form_1 \vee form_2$, $\neg form_1$, and $(form_1)$ are formulas, with their standard meanings.

3. If *form* is a formula with a free variable s, then $(\exists\ s)(form)$ is a formula.

In addition, there are the usual rules for guaranteeing that formulas are safe, i.e. that queries don't ask for objects that are not in the database.

4.2. Examples of Queries

Query 1: Find shipments of gasoline to Stockholm whose ship report says "No whale sighted."
{entity u : u is of type Shipment $\wedge$ dest(u) is of type geographical region $\wedge$ dest(u) = Stockholm $\wedge$ cargo(u) = gasoline $\wedge$ report(u) has keyword "No whale sighted."}

Query 2: Find shipments of petroleum products to Sweden. (Note that this should return entities found in the first query.)
{entity u : u is of type Shipment $\wedge$ dest(u) is of type geographical region $\wedge$ dest(u) is a subvalue of Sweden $\wedge$ cargo(u) is a subvalue of petroleum product}

Query 3: What has Sweden as a property?
{entity u : $\exists$ f $\wedge$ f(u) = Sweden}

Query 4: What types does Sweden belong to?
{type t : Sweden is of type t}

4.3. User Interface for Retrieval

Since the explorer may have no knowledge of the schema, he or she builds up a query as a boolean combination of *probes*, each built by stepwise specialization.

The environment of a given probe is a triple consisting of a type t and a set of function-value pairs fv

[4] - A partial order $\leq$ has the property that $\forall$ a, a $\leq$ a and $\forall$ a, b, and c, a $\leq$ b $\wedge$ b $\leq$ c $\rightarrow$ a $\leq$ c. The order is partial because every two distinct elements need not be comparable.

[5] Every tuple of a conventional database is coherent in this sense, where the reader's knowledge consists of understanding the schema of the relation containing the tuple.

[6] - We also allow aggregate queries of the form agg unique ({target : qualification}) or agg all ({target : qualification}), where unique and all indicate removal and non-removal of duplicates from the collection of targets.

$(f_1,v_1), \cdots, (f_j,v_j)$ of functions for which some values have been defined, and a set of entities E of type t that have been defined by fv. We denote this triple by (t,fv,E).

The commands all refer to this environment. For example, one command causes the system to present the subtypes of the current type t. Another presents the functions applicable to t.

In response to each command, the system constructs a menu dynamically. The construction strategy makes use of the subvalue and subtype partial orders. For example, if the explorer partially specifies the destination function, dest(u) = Europe, then asks to specify the destination further, the system presents the maximal subvalues of Europe, {British Isles, Scandinavia, ... }. The system presents disjoint maximal subvalues to avoid redundancy. So, it wouldn't present {British Isles, Scandinavia, Sweden ...}.

5. Data Model: Presentation

Most database management systems provide facilities to order sets of tuples. These sorting facilities are not of much research interest. They are, however, of interest to a knowledge exploration system for two reasons:
1) coherent units and their properties may contain a lot of text (by contrast, tuples contain little text);
2) an explorer may have to read and understand background (prerequisite) information to understand a given coherent unit (by contrast, tuples such as (John Smith, 20000 dollars) are understandable in isolation of other tuples).

The first point implies that remembering a coherent unit takes effort. The second implies that there are relationships among the entities retrieved by a query that the explorer may not know about. The system should use these relationships in its presentation strategy. Our strategy is based on the idealized assumption that a person's effort is proportional to the number of coherent units he or she must remember.

5.1. Fragment Theory for Presentation

This part of fragment theory is concerned with the effort required to understand a presentation. We motivate our effort measure through two examples.

Suppose the explorer wants to compare entities A and B (say two concurrent programming languages) based on functions f, g, and h. Consider the presentation orders P = (f(A), f(B), g(A), g(B), h(A), h(B)) and P' = (f(A), g(A), h(A), f(B), g(B), h(B)). In the first presentation, the user must remember f(A) only as long as it takes to read f(B). In the second, the user must remember f(A) while reading g(A), h(A), and f(B). Thus, P' requires the user to remember four coherent units, whereas P only requires the user to remember only two at a time.

As a second example, consider the prerequisite relationships (see figure 2) {(A,F), (B,F), (C,F), (D,G), (E,G), (F,H), (G,H)}. Consider the two presentations: P = (A, B, C, D, E, F, G, H) and P' = (A, B, C, F, D, E, G, H). Assuming that an explorer cannot understand a coherent unit without remembering all its immediate prerequisites, the presentation P requires the user to remember A through E while reading F, whereas P' requires the user to remember only A, B, and C while reading F, then only F, D and E while G is read. This time, P requires more effort than P'.

5.1.1. Measure of Effort

Weighted pebbling problem - We construct a (possibly cyclic) directed graph called a *presentation graph* $G=(V,E)$ as follows: the vertices are the coherent units retrieved by a query. There is a directed edge from v to v' if either v is a prerequisite of v' or the explorer has established some ordering or grouping relationship between v and v'.

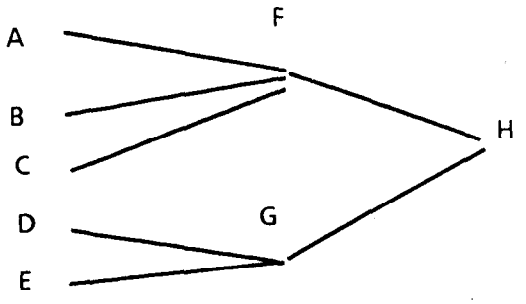

Figure 2: Prerequisite graph. Prerequisite edges directed from left to right. Presentation (A, B, C, D, E, F, G, H) requires six pebbles, whereas (A, B, C, F, D, E, G, H) requires only four.

Treat each maximal strongly connected subcomponent as a supernode whose weight is the number of nodes in that subcomponent. This yields a *reduced graph* on the supernodes, $G'=(V',E')$, where we denote the weight of supernode n as $w(n)$. (The reduced graph is acyclic, because the prerequisite relation is acyclic.) A supernode n is *fully pebbled* if it has $w(n)$ pebbles on it. A pebble may only be placed on a supernode if all the supernode's immediate predecessors are fully pebbled. No pebble may be placed on a supernode after a pebble is removed from it.

Suppose G is a presentation graph with associated reduced graph G'. A presentation P has cost k for graph G if it legally pebbles G' with no more than k pebbles under the interpretation that when node n appears in P, a pebble is placed on the supernode associated with n in G' and pebbles are removed when they are no longer needed.

In the first example, P had cost two whereas P' had cost four. In the second example, P had cost six, whereas P' had cost four.

There is a natural optimization problem associated with presentation graphs. The *progressive pebbling problem* is, given a presentation graph G, to find the presentation having the lowest cost for G. Sethi [Seth75] has shown this problem to be NP-complete for directed acyclic graphs with nodes of uniform weight.

5.2. Presentation in our Data Model

Since different presentations may entail different effort, we want the explorer to state his or her preferences. Since the relationships may not specify a total order, the system should then try to optimize the presentation to minimize effort.

5.2.1. Linguistic facilities

The system allows the user to specify a partial ordering on a set of coherent units of the same type. There are two possibilities. The first is to specify a sequence, thus a total ordering. The second is to specify the partial order as a directed graph. For example,
Order E by {(x,y) : (dest(x) is a subvalue of Sweden and dest(y) is not a subvalue of Sweden)} puts the coherent units with destination Sweden before any others.

The system also allows the explorer to group entities into equivalence classes. Different equivalence classes may be displayed in any order, but the entities in a single class should be displayed together. For example,
Group E by {{x,y} : dest(x) = dest(y)}.

The second group of facilities allows the user to juxtapose entities and their functions in various ways.

Compare (partial order or grouping on entities, sequence of functions). This statement causes the system to list the values of the entities function by function according to the function sequence. For example, compare((Ada, Concurrent Pascal, Simula), (synchronization mechanism, concurrency expression)) would list the synchronization mechanism of each language, then the method of expressing concurrency of each language.

Display(partial order or grouping on entities, sequence of functions) This causes the system to list each entity together with the values of the functions associated with it.

5.2.2. System optimizations

The system constructs a presentation graph from the ordering information.[7] Then the system adds prerequisite edges that don't contradict the ordering information (i.e. don't add A → B if B → A was specified in the ordering). Finally, the system tries to find the lowest cost presentation.

Since the optimization problem is NP-hard, we look for heuristics based on subproblem with optimum solutions. One such subproblem is a tree (e.g. figure 2) whose prerequisite edges point towards the root.

The recursive strategy for such a structure T is to call pebble on the root:

```
procedure pebble(T,r)
begin
 if r has at least one child
   then
     pebble the subtrees rooted at the children of
          r from the most expensive to the least expensive,
          leaving pebbles only on the roots of the subtrees;
 pebble r
end
```

To calculate the cost of pebbling a root, use the following recursive function.

```
function cost(T,r)
begin
 if r has at least one child
  then
   begin
    let c_1, ..., c_k be the children,
     such that cost(T,c_1)-w(c_1) ≥ ... ≥ cost(T,c_k)-w(c_k);
    return w(r) + max(cost(T,c_1), cost(T,c2) + w(c_1),
           ... , cost(T,c_k) + w(c_1) + ... + w(c_{k-1}))
   end
    else return w(r)  {the weight of node r}
end
```

In the example of figure 2, the subtree rooted at F should be pebbled first, because it has cost four, whereas the subtree rooted at G has cost three. The cost of pebbling the entire tree is also four.

[7] - Different nodes of this graph may be the same coherent unit. For example, two languages being compared according to their synchronization mechanism may have the same one.

The heuristic we use is expressed in the following procedure, (here, G is a reduced graph):

```
procedure heurpebble(G)
 begin
  for each sink s in the reduced graph,
   do begin
        consider the subgraph consisting of s and
        its predecessors;
        remove edges randomly from that subgraph until a
         tree T(s) remains containing all predecessors;
    end;
   Pebble each tree according to procedure pebble,
    but don't remove pebbles from nodes that are sources
    of non-tree edges whose sinks are not fully pebbled.
    Also, don't add pebbles to fully pebbled nodes
    or nodes corresponding to the same coherent unit
    as some pebbled node.
end
```

For example, if there were an edge from A to G in the graph of figure 2, we would pebble F's subtree before G's subtree as before, except that we wouldn't remove the pebble from A even after F was pebbled. This would increase the cost to four.

6. Heuristic enhancements from artificial intelligence

Our data model assumes that manipulating or processing text within a coherent unit is impossible. We make that assumption to remain independent of the natural language understanding problem. However, certain manipulations may be possible.

For example, coherent units may not normally contain anaphoric references (such as pronouns) to other coherent units. This is to avoid having an explorer read a coherent unit containing "He did X." and not know whom "He" refers to. In certain cases, the explorer would know, say, that the referent of "He" is "John Smith". In those cases, the system should substitute "He" for "John Smith". To implement this, one might use rule-driven system of the form: (if user came from any one of these coherent units ..., then replace "John Smith" by "He" in these coherent units ...).

A second enhancement is to have the system try to acquire a model of the explorer and then to present coherent units at an appropriate level of difficulty for him or her. To do this, there would have to be a specially treated function conveying level of difficulty, which the system would use. Another possibility is to include a summarization relation in the data model, thus permitting an explorer to get either a long or a short explanation of a topic.

A third enhancement is to try to infer connections between coherent units unforeseen by authors. Robert Amsler and Donald Walker in an unpublished study have found that by assembling the possible topics corresponding to each keyword in a paragraph of a New York Times article, a program that chooses the topic most frequently mentioned usually arrives at the topic of the article. Besides helping authors, this inference mechanism may help explorers find coherent units related to ones they have already explored.

7. Example: this article as a NetBook

A recipe for constructing a NetBook from a standard article is first to construct an outline (figure 3) of the major points. The headings become the function names whose domain is the coherent unit containing the main idea of the

article. Second, construct other groupings from the same coherent units if that seems appropriate. These may become other function names. For example, new facilities is an added function whose range is decomposition, subvalue, autonomous ordering based on prerequisites, and linguistic support for presentation. Third, add connections. There are prerequisite connections due to definitions: information unit, coherent unit, irrelevance, redundancy, pebbling model. There are motivational connections: from the subvalue facility in data model to the soft aggregate section, from the compare construct to the example of comparison, and from the prerequisite algorithm to the example of figure 2.

Using these connections, we can directly answer questions such as "Why is there a subvalue facility?" or "Tell me about the new facilities." We could also answer broader questions such as "What is the redundancy concept and how does it influence the rest of the paper?" To answer this question, we would form a query that considered the connections to the redundancy concept: the example of redundancy in linear text, the grouping and predicate selection problems, decomposition, and soft aggregates. If this article were part of a larger collection, we might want to compare aspects of this data model with other models. In that case, we would have to break up the retrieval portion of the data model into smaller coherent units.

8. Conclusion

The knowledge explosion suggests a need to build and study systems that make the job of the knowledge explorer easier. Historically, the designers of information retrieval and natural language understanding systems have assumed that the writing process is separate from the exploration process and have designed systems to bridge the gap using electronic and human intermediaries. Hypertext, database management, and computer-aided instruction systems provide an alternative model in which the knowledge creator tries to put data into a form that is convenient to explorers. The NetBook data model follows this alternative approach.

The novel facilities of our data model are a decomposition criterion, the subvalue partial order, linguistic facilities for ordering coherent units, and an autonomous presentation strategy based on prerequisites. Fragment theory provides the rationale for these facilities. Its usefulness stems from its simplicity. Our confidence in it (despite its seeming arbitrariness) stems from its ability to model rhetorical paradigms used by journalists for ordering information and software engineering paradigms such as information hiding. It is not yet a design theory for knowledge exploration systems, but we think it will be.

We are constructing a prototype to support the NetBook data model. We expect it to help us study aspects of the user interface such as the effect of screen resolution, heuristic enhancements such as those of section 6, and facilities for supporting authors. This last problem is particularly important for psychological reasons; people have been so well trained as writers to put ideas in a linear order that we think it's easier.

netbook as idea (netbook entity)
  abstract, conclusion
  books vs. people

motivation

related work
  information retrieval
  database
  natural language processing
  intelligent computer aided instruction
  hypertext

fragment theory for retrieval
  information unit
  linear text
  measures of redundancy
  grouping selection
    formalization as a problem
  predicate selection

fragment theory for presentation
  comparison example
  prerequisite problem
  presentation graph formulation

data model
  retrieval - structures, operations, interface
  presentation - ordering, compare, prerequisite

new facilities
  decomposition
  subvalue
  autonomous ordering based on prerequisites
  linguistic support for presentation

ai heuristics

example
the present section.

Figure 3: The coherent units of this article.

References

B82 - J. S. Brown, "Notes Concerning Desired Functionality, Issues and Philosophy for an AuthoringLand" CIS working paper, Xerox Parc, July, 1982.

BM85 - D. C. Blair and M. E. Maron, "An Evaluation of Retrieval Effectiveness for a Full-text Document-Retrieval System," Communications of the ACM March 1985 vol. 28, no. 3 pp. 289-299

BRB76 - J. S. Brown, R. Rubinstein, and R. Burton, "Reactive Learning Environment for Computer-Assisted Instruction" BBN Report No. 3314, Bolt Beranek and Newman, Inc. Cambridge, Mass. 1976.

B45 - V. Bush, "As we may think" Atlantic Monthly, 176, 101-108. July, 1945

C76 - A. Collins, "Processes in Acquiring Knowledge" in *Schooling and the Acquisition of Knowledge* R. C. Anderson et al. (Eds.) Hillsdale, N.J.: Lawrence Erlbaum, pp. 339-363.

C80 - R. G. G. Cattell, "An Entity-based Database User Interface" Proc. of 1980 ACM-SIGMOD Conference on Management of Data, May, 1980.

CDFR83 - A. Chan, U. Dayal, S. Fox, and D. Ries, "Supporting a Semantic Data Model in a Distributed Database System" Proceedings of the Very Large Data Base conference 1983, pp. 354-363.

D83 - U. Dayal, "Processing Queries over generalization hierarchies in a multidatabase system," Proceedings of the Very Large Database conference pp. 342-353 1983

E84 - D. C. Engelbart, "Collaboration Support Provisions in AUGMENT" AFIPS 1984 Office Automation Conference, Los Angeles.

EWN73 - D. C. Engelbart, R. W. Watson, and J. C. Norton, "The Augmented Knowledge Workshop" Proceedings AFIPS National Computer Conference, 42 (Arlington, Virginia), pp. 9-21, 1973.

F84 - D. Fogg, "Lessons from a 'Living in a Database' Graphical Query Interface," ACM-SIGMOD Conference on Management of Data, 1984, pp. 100-106.

GHF82 - R. Grishman, L. Hirschmann, and C. Friedman, "Natural Language Interfaces Using Limited Semantic Information," COLING 82: Proceedings of the Ninth International Conference on Computation Linguistics (J. Horecky, ed.) pp. 89-94, North-Holland, Amsterdam.

GJ79 - M. R. Garey and D. S. Johnson, *Computers and Intractability -- a guide to the theory of NP-completeness* W. H. Freeman and Company, 1979.

H80 - C. F. Herot, "Spatial Management of Data" ACM Transactions on Database Systems 5 (4), December 1980.

LM79 - H. Levesque and J. Mylopoulos, "A Procedura Semantics for Semantic Networks" in *Associative Networks* N. V. Findler (ed). Academic Press, New York.

L85 - D. Lowe, "Cooperative Structuring of Information: the Representation of Reasoning and Debate" International Journal of Man-Machine Studies (to appear in 1985).

MM79 - M. M. Mantei and D. L. McCracken, "Issue Analysis with ZOG, a Highly Interactive Man-machine Interface" First International Symposium on Policy Analysis and Information Systems, June 1979.

N79 - N. Negroponte, "Books Without Pages", 1979 IEEE International Conference on Communications IV, Boston, Massachusetts.

Nel83 - T. H. Nelson, *Literary Machines* Box 128 Swarthmore, Pennsylvania, 1983.

S70 - G. Salton, "Automatic text analysis" Science 168, 3929 April 1970, pp. 335-343

S81 - N. Sager, *Natural Language Information Processing: A Computer Grammar of English and its Applications* Addison-Wesley, Reading, Mass.

S85 - D. Shasha, "Netbook -- a data model to support knowledge exploration" Technical Report, Courant Institute, New York University, 1985.

SA77 - R. C. Schank and R. P. Abelson, *Scripts, Plans, Goals and Understanding* Hillsdale, N. J. : Lawrence Erlbaum, 1977.

S75 - R. Sethi, "Complete Register Allocation Problems," SIAM Journal on Computing 4, 1975, pp. 226-248.

Ship81 - D. Shipman, "The functional data model and data language Daplex" ACM Transactions on Database Systems, vol. 6, no. 1, March 1981.

So84 - J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine* Addison-Wesley, Reading Mass, 1984.

SCG78 - A. L. Stevens, A. Collins, and S. Goldin, "Diagnosing Students' Misconceptions in Causal Models," BBN Report No. 3786, Bolt Beranek and Newman Inc. Cambridge Mass., 1978.

SK82 - M. Stonebraker and J. Kalash, "TIMBER: A Sophisticated Relational Browser" Technical Report, University of California Electronics Research Laboratory, May, 1982.

T83 - R. H. Trigg, *A Network-Based Approach to Text Handling for the Online Scientific Community* Ph.D. Dissertation, Department of Computer Science, University of Maryland, 1983.

U82 - J. D. Ullman, *Principles of Database Systems* Computer Science Press, Rockville Maryland, 1982.

VJ84 - Y. Vassiliou and M. Jarke, "Query Languages -- a taxonomy," in *Human Factors and Interactive Computer Systems* Y. Vassiliou, Ed. Ablex, Norwood, N.J. 1984.

WH81- D. E. Walker and J. R. Hobbs, "Natural Language Access to Medical Text," Technical Note 240, SRI International, March, 1981.

Wey82 - S. A. Weyer, "Dynamic Book for Information Search" International Journal of Man-Machine Studies, vol. 17, pp. 87-107, 1982.

ZD69 - P. Zunde and M. E. Dexter, "Indexing consistency and quality," Americant Documentation 20, 3, July 1969, pp. 259-264