Search and Preference-Based Navigation in Electronic Shopping

April 1, 1993

Tomás Isakowitz
Department of Information Systems
Stern School of Business
New York University
44 W. 4th St.
New York, New York, 10012
tisakowi@stern.nyu.edu

Steven O. Kimbrough
Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104

kimbrough@wharton.upenn.edu

Working Paper Series
STERN IS-93-9

(replaces: IS-90-8)

Search and Preference-Based Navigation in Electronic Shopping

Abstract

The aim of this paper is to address the requirements for electronic shopping systems. Large-scale computerized electronic shopping systems need to accommodate both (a) a large number of products, many of which are close substitutes, and (b) a heterogeneous body of customers who have complex, multidimensional—and perhaps rapidly changing—preferences regarding the products for sale in the system. Further, these systems will have to be designed in a manner so as to both (c) reduce the complexity of the shopping problem from the customer's point of view, and (d) effectively and insightfully match products to customers' needs. We show how an abstraction hierarchy with an imposed distance metric provides the necessary elements to implement the desired features. Further, we indicate how the distance metric, in the context of the abstraction hierarchy, can be interpreted as a unidimensional utility function. Finally, we extend the single dimensional (single perspective) treatment to multiple dimensions, or *perspectives*, and show how the resulting representation can be interpreted as a multiattribute utility function. We argue that the resulting function is plausible and, most importantly, testable.

Keywords: decision analysis, decision support systems, electronic shopping, preference modeling, user interfaces, utility theory, multiattribute utility theory.

1 Introduction

Technical developments and economic forces are evolving in a direction favoring computer- and communications-based services for purchasing activities, either by consumers or by businesses. On the technical side, personal computers and workstations continue to become more powerful, and to have increasingly sophisticated software; communications networks continue to proliferate, and the infrastructure to support them—including fiber optics transmission facilities and network services such as ISDN—continues to be developed at a rapid pace. On the economic side, computing and communications continue to become cheaper; time and labor continue to become more expensive; markets continue to expand both in the variety of products offered and the reach of the companies offering these products, viz. 24-hour, worldwide trading of securities; and globalization of commerce continues to accelerate. While the adoption and eventual impact of electronic shopping systems is uncertain, it is crucial at this stage to investigate the theory and design principles for supporting such systems.

In the presence of large databases of products, e.g., J.C. Penney's, Land's End's or L.L. Bean's catalogs, customers need guidance and advice while shopping. Moreover, if we consider the whole consumer marketplace as an arena for electronic transactions, we soon encounter complexities in searching through such a vast information space. The ability to support search in these large information bases will have a significant impact on the success of electronic shopping systems.

The customer should not face the burden of finding the right item by running through screen after screen of product descriptions. A good system will play the role of a good salesperson by understanding what the customer is looking for, remembering his or her personal preferences and making appropriate suggestions. In addition, a system should represent the seller's interests by offering profitable items.

Our aim in this paper is to propose implementation principles for such a *salesperson* system. We suggest using artificial intelligence techniques to achieve the desired results. Our work extends Lee and Widmeyer's [9] ideas on using a graph representation to guide the search process. We use multiple graphs to represent different search perspectives. Moreover, we propose an *algebra of graphs* that supports the combination of various search dimensions. For example, a shopper should be allowed to search not only for a pair of pants (type of clothing perspective), but should also be able to specify that she wants it blue (color perspective) and light in weight for the summer (season perspective). Further, we are able to interpret our representation as an encoding of a multiattribute utility function (see §8). This permits the theoretical apparatus of utility theory to be brought into play in order to validate any particular representation in a given application.

We also deal with two basic problems: *surplus* and *shortage*. *Surplus* occurs when the customer issues a vague request. If, for example, she asks for a pair of pants, the system has to narrow down the search so that it can come up with a good candidate. In order to do this it might take into account other information such as cost, season, color preferences,

the sex of the person that will use it, and so forth. *Shortage* occurs when a request is issued for an item which is not available. Suppose that a customer asks for item number *jcp 279-1730 d*—a pair of pants—that is currently out of stock. The system should offer a reasonable substitute. This means a pair of pants of similar color, cost-range, fashion category, and so on.

There is a rich and broad range of issues to be taken into account when designing and developing systems to support electronic shopping. There are marketing issues, system issues and strategic issues. In this paper, our concern is with the problem of *what* information should be presented to electronic shoppers, rather than with *how* that information is presented (i.e. content rather than format.) Thinking in terms of decision support systems (DSS), our focus is on the problem processing and the knowledge subsystems for an electronic shopping DSS [2].

The remainder of the paper is organized as follows. We begin, in §2 by discussing electronic home services and factors relevant to their acceptance. In §3 we continue with a discussion of Lee and Widmeyer's work, which encompasses only a single perspective (single attribute, unidimensional) case. In §4 we extend this framework by elaborating upon its basic data structure. §5 extends our treatment of the problem to the multiple perspective case. We show, in §6, that the shortage, as well as the surplus, problem can be handled in essentially the same manner under our representation scheme, and in §7 we discuss how learning and idiosyncratic preferences can be incorporated into our proposed system. We demonstrate,

in §8, how the measure of preference we have used throughout is an implicit utility function and we present certain features of this function, which could be used in developing a valid representation. We conclude in §9.

2 Delivering Electronic Services to the Consumer Market

In 1992, the videotex market is forecasted to account for $700 million in revenues [11]. Videotex is "the delivery of electronic information to the consumer market [6]." Prodigy and Compuserve are the two best known videotex systems in the United States. Minitel's success in France has been attributed in part to the government's backing. Included in their services such as stock prices, travel information, specialized news, these systems offer various kinds of electronic shopping services. As videotex systems expand their customer base, the demand for electronic shopping will increase. Of the 85 million households in the U.S., 25 million have a personal computer [6]. Although only a small fraction of these– 1.5 million– actually subscribe to online services, the growth potential for this industry is fabulous.

Although videotex was initially pronounced a failure [10], a number of lessons learned from the early experience guides the deployment of new services. One of the key lessons is that "information content determines the success of a service [5]." The systems will be successful as long as they deliver information in cheaper and less expensive ways than those otherwise

available. For instance, consumers will not pay a premium for information (market prices, entertainment listings, etc.) that is available in newspapers. A second lesson indicates that the systems should be entertaining and easy-to-use. One of the causes for Minitel's success is its is easy to use interface [5]. Videotex systems should not require special equipment. Since customers resent tying up household equipment such as the TV, it makes sense to use the existing base of personal computers to deliver the service.

The advantages of electronic shopping systems are:

- ability to provide continually updated, current information
- potential for substantially lowering shoppers' transaction costs [3, 15, 16], including both the costs associated simply with ordering and invoicing, and the costs associated with searching among offerings in a market
- opportunities in the implementation of new marketing strategies (advertisements can be done via the system) and in measuring their effects.

It is important to entice customers so that they prefer shopping electronically to doing it in stores or by telephone. Since friendly trained telephone operators can offer help and guidance, the electronic system should offer comparable services. In this paper we propose the use of intelligent search mechanisms to provide the electronic equivalent of a friendly salesman, who guides and advises customers based on his own experience and on the preferences expressed by the customers.

3 Previous Work

Our purpose in this section is to review earlier work on this subject by Lee and Widmeyer [9] and to indicate ways in which we propose to extend it. Unless otherwise noted, the proposals and data structures we discuss in this section are those originally presented in [9].

The general problem is how to organize the information on the items for sale in a way that can effectively support a search process. Printed catalogs tend to classify the items for sale into categories. To locate a desired item, customers determine the category to which it belongs and proceed to browse through it. For example, a catalog for a clothing shop splits the items into pants, shirts, jogging suits, swimming suits, gloves, coats, and so on. Furthermore, it seems natural to group coats and gloves together since both are outdoor garments, and jogging and swimming suits could be grouped together as sportswear. In this manner a hierarchical organization for representing the products for sale is reached.

Lee and Widmeyer [9], using ideas from semantic networks, propose representing the structure as a directed graph, as shown in Figure 1. The nodes are classification categories, the leaves are individual items, and the directed arcs represent inclusion of categories and are called *isa-links*. The items for sale are *gloves, coats, jogging suits, swimming suits,* and so forth. The nodes *clothes, outerwear, daywear, sportswear* and *pants* represent abstract categories. These are groupings used to support the search process. The leaves in the

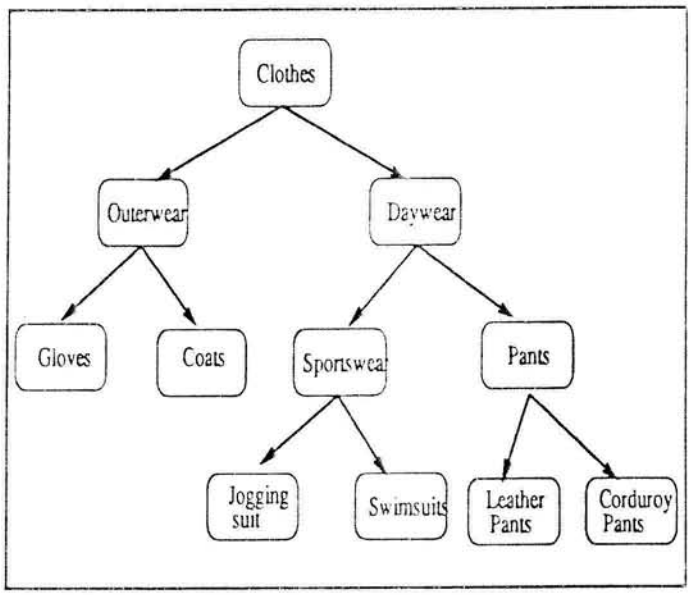

Figure 1: A Graph for Clothes

graph represent actual individual product units for which additional information on the quantity on stock is kept. The graph imposes a natural distance notion, as the number of arcs between two nodes. For example, *jogging suits* are closer to *swimming suits* (2 arcs away) than to *leather pants* (4 arcs away). A request for an item of a category is handled by performing a graph search that arrives at the *closest* item. For example, in response to a request for a sportswear item, jogging suits or swimming suits are proposed prior to proposing leather pants.

In [9], Lee and Widmeyer outline a Prolog implementation. The *isa* links are represented by predicates: `isa(Cat1,Cat2)`. For example, the leftmost path in the graph of Figure 1 is represented by the following predicates:

```
isa(gloves, outerwear).
isa(outerwear, clothes).
```

Stock availability of the leaf nodes is indicated by assigning codes to physical entities. The `instance_of(Individual, Category)` predicate is used for this purpose. For example, `instance_of("jp 279-1730 d", "corduroy pants")` represents the fact that the product with code `jp 279-1730 d` is a pair of corduroy pants. Although these codes could be included in the graph itself, this would clutter the graphs and duplicate information –in the cases where an item belongs to more than one category.

Although all graphs considered by Lee and Widemeyer are, in fact, trees, their method also supports *multiple inheritance* links, i.e. graphs which have nodes with multiple parents. A node x is such a node if there are nodes y_1 and y_2 such that

1. $y_1 \neq y_2$; and
2. $isa(x, y_1)$, and $isa(x, y_2)$.

The predicate `match` is used to find suitable candidates for a user's request. (We comply with Prolog's notation where variable names start with an uppercase letter.)

```
match(Item, Individual) :- instance_of(Individual, Item).
match(Item, Individual) :- isa(Desc, Item),
```

```
match(Desc, Individual).
```

The recursive definition of this predicate provides a simple, yet correct method for finding matches within a subgraph rooted at a node. It searches through the descendants of a node in order to find an individual of the desired category. For example, using the clothing graph presented in Figure 1, a request for a `sportswear` item, would launch a graph traversal that first checks availability *jogging suits*, and then, if none are available, for *swimming suits*. The search fails if no items are available in any of these categories. A natural extension of this method is to continue the search in a sibling subtree, in this case, the subtree rooted at *pants*. Thus the system might propose leather pants, that, although not a sportswear item, may represent the best match the store can offer. This is implemented via the predicate `pmatch`:

```
pmatch(Item, Individual) :-   match(Item, Individual).
pmatch(Item, Individual) :-   isa(Item, Parent),
                              pmatch(Parent, Individual).
```

Although an interesting and useful approach to the problem, this graph representation and Prolog program are not sufficiently powerful to handle certain aspects of the salesperson's role. First, the graph search is dependent upon the order in which the Prolog predicates are written. In the previous example when looking for a match for *sportswear*, *jogging*

suit will be tried before *swimming suit* if the predicate `isa(jogging suit, sportswear)` appears before `isa(swimming suit, sportswear)` among the Prolog facts. Thus the only way of indicating preferences between siblings is via the order of the predicates, a rather undesirable and subtle feature. Note that this is due to the absence of mechanisms for specifying preferences among siblings in the graph. Secondly, Lee and Widemeyer [9] handles shortage and surplus differently. This results in different algorithms, and –to some extent– different data structures. Since both are instances of a search problem, a uniform treatment of both situations is desirable. Most significant however, is our last point: in general, there is more than one aspect to take into account when taking care of a customer. For example, when buying clothes it is not only important to consider the categories as described in Figure 1, the system should also consider other constraints such as cost, color, fabric, season, personal preferences, etc. In this paper we propose an expanded data structure and new operators to address these issues.

4 Expanding the Graphs

In this section, we augment the graphs by assigning costs, or weights, to the arcs. These costs are used to penalize the choice of some nodes over others. As we shall see, this solves some of the problems presented in the previous section. We also discuss a prototype implementation of the data structure and new algorithms.

4.1 Assigning costs to the arcs

In order to specify a more precise notion of distance between nodes in the graph, we assign costs to the arcs. This is done via a *cost-assignment* function that maps arcs to costs. The cost of a path is the sum of the costs of its arcs. We define the distance between two nodes as the minimum over the costs of all paths connecting them.

Intuitively, low cost should be equated with close similarity. This however, is dependent upon the right choice of a cost assignment function. The system we propose is very sensitive to variations in the cost figures. Hence detailed attention should be provided to the cost assignment function. Although this problem is itself quite interesting, an in-depth treatment is beyond the scope of this paper. In section §8, we describe how to use the utility model [12] to validate cost-assignments. Prospect theory [14] is another approach to this problem; and Thaler [13] compares both approaches. The focus of this paper is not on determining appropriate cost assignment functions, but on managing and using these graphs to support electronic shopping once an appropriate cost assignment function has been applied to construct the extended graph. In what follows we will assume the existence of a cost assignment function.

The cost of the arc from a parent node, i, to one of its child nodes, j, will be denoted by $C_{i,j}$, and will be, without loss of generality, a number in the interval $[0, 1]$. Figure 2 presents a cost assignment applied to the `clothes` graph. Note that the cost of the arcs connecting

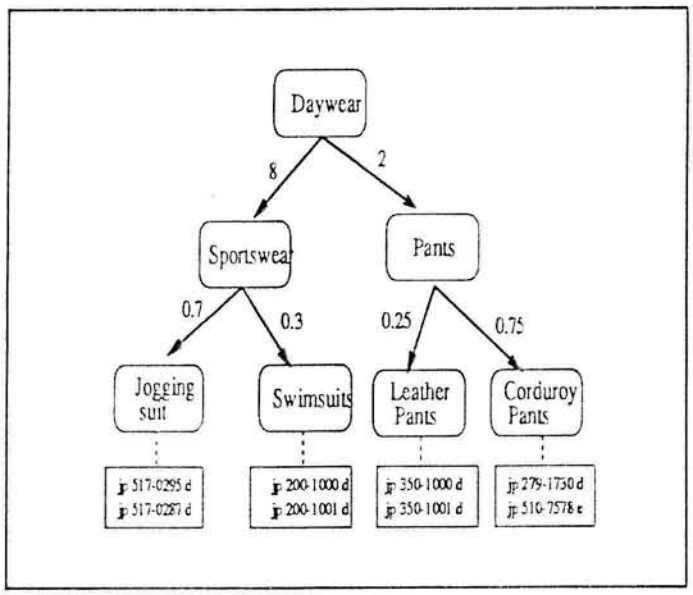

Figure 2: Adding Costs to the Arcs

`daywear` to `pants` and to `sportwear` are not in the interval $[0, 1]$; they have been scaled as we explain later. The arc costs represent an intentional bias: `leather pants` are preferred to `corduroy`; and `swimsuits` to `jogging suits`. This bias is a consequence of the cost assignment function, and it might reflect consumer preferences, or company policy (favor sales of some higher margin items over others). The codes (catalog numbers) appearing inside the rectangular boxes correspond to individuals `instance_of` predicates. They are included for illustrative purposes only. Throughout this and the following sections we will use that subgraph of Figure 2 as our example, rather than the whole *clothing* graph (Figure 1), to simplify and shorten the discussion.

A potential difficulty with arbitrary cost-assignment functions is that they may not respect

the hierarchy of the graph, in fact they may obliterate it. For example, nothing prevents the cost of the path from `sportswear` to `corduroy pants`, to be less than the cost of the path from `sportswear` to `swimsuit`. This is problematic because the rationale beyond the introduction of the cost-assignment functions is to refine the distance notion induced by the graph. Therefore, the cost-assignment distance should *extend* the graph-induced distance, and hence, be consistent with it. In order to overcome this difficulty, we propose the following principle:

> *Let $Below(N)$ be the set of all nodes that are connected to N and are* below *N (if these graphs were trees, this set would be the set of descendants of N). The total cost of a path connecting any two nodes in $Below(N)$ should always be less than the cost of connecting a member of $Below(N)$ with another node that is not in this set.*

This can be achieved by scaling the costs according to the *level* of a node. The *level* of a node is defined as the length of the longest path connecting it to a leaf. In Figure 1, the level of `daywear` is 2, and the levels of `sportswear` and `pants` are both 1.

We will show how to choose a scaling factor so that arcs coming into nodes at the level of N are assigned a cost higher than the cost of any path within $Below(N)$. This will enforce the desired principle. To see this, consider a path passing through N that connects a node in $Below(N)$ to a node outside the set. By the restriction on the kinds of graphs we use,

there has to be an arc in this path that is coming into N or into another node N' with the same *level* as N. By construction, the penalty for traversing this arc is higher than the entire cost of any path connecting two nodes in $Below(N)$. Hence, the desired property is satisfied.

The scaling factors for arcs coming into a node of level l are of the form k^l. The constant k has to be chosen to satisfy the above stated principle. Let m be the number of nodes with more than one parent in the graph (if the graph is a tree, then $m = 0$). For any level l, the constant k has to satisfy

$$k^l > (m+1)2\sum_{i=0}^{l-1} k^i \tag{1}$$

The sum $\sum_{i=0}^{l-1} k^i$ is an upper bound for the cost of a path connecting a leaf node to a node of level l. It is multiplied by 2 so that the path can go up to N and down to a leaf. It is multiplied by $m+1$ to accommodate instances of multiple inheritance. Since k and m are greater than 1, this results in the following constraint for k:

$$k^l > (m+1)2\frac{k^l - 1}{k-1} \tag{2}$$

which is equivalent to

$$k^{l+1} - (2m+3)k^l + 2(m+1) > 0 \tag{3}$$

Clearly, if this inequality is satisfied for $l = 1$, it will be satisfied for $l > 1$. Hence it is enough to choose k so that:

$$k^2 - 2(m+3)k + 2(m+1) > 0 \tag{4}$$

Since m is positive, we obtain

$$k > \frac{(2m+3) + \sqrt{4m^2+4m+5}}{2} \tag{5}$$

In the clothing example, $m = 0$, hence k can be chosen as a number greater than $\frac{1}{2}(3+\sqrt{5}) = 2.618$. We have picked, arbitrarily, the value $k = 10$. In Figure 2, the costs of the original *clothing* graph have adjusted using the scaling factor 10^l.

Node	Distance
Leather Pants	*0.25*
Corduroy pants	*0.75*
Swimming suit	*10.3*
Jogging suit	*10.7*

Table 1: Calculated scaled distances from *Pants*

Let us analyze via an example how the scaled costs influence the distance measurements. Table 1 shows the distances from *Pants* to all leaf nodes. We have included only leaf nodes because it is among those nodes that the system makes its suggestions. It would not make sense to suggest an abstract category such as *Clothes* or *Daywear*. The nodes in the table are ordered from closest to furthest, thus the first suggestion is *Leather pants*, then *Corduroy pants*, and so forth. Notice that we have succeeded in specifying a preference between siblings. Only if the system is indifferent to a choice between two items, will they have the same cost, in which case, the system should suggest both to a customer.

The use of costs in the graphs, opens the doorway to a number of applications. For example,

using the same underlying graph structure, different personal preferences can be specified by changing the costs of certain arcs. This will be explored in section 7. Similarly, if the management wants to stress a product over others, it can do this by lowering the costs associated with the corresponding nodes. As we can see, the use of cost assignments increases the accuracy and flexibility of an electronic shopping system.

Note that the above described constructs apply not only to trees, but to graphs with multiple inheritance as long as *levels* can be safely assigned to nodes the notion of *level of an node*, that is, the graphs are acyclic. Throughout the rest of these paper, we will assume that all graphs are of this kind.

4.2 Modifying the Data Structure

The implementation of the expanded graph as a data structure is straightforward. An argument representing the cost of the arc is added to the `isa` predicate. Since the system will deal with more than one graph, we also add an argument for the graph name. The predicate becomes: `isa(Graph_name, Child, Parent, Cost).` As before, the actual leaves are represented via the `instance_of` predicate. There are no costs associated with these predicates because they are not part of the hierarchy. A portion of the graph depicted in Figure 1 is represented by the following predicates:

```
isa(clothing, leather pants, pants, 0.25).
```

```
isa(clothing, corduroy pants, pants, 0.75).
```

4.3 A note on complexity issues

Recall that the distance between two nodes is defined as the cost of the shortest path connecting them. Fortunately, reasonably fast algorithms are available to compute shortest paths. There are at least two possible implementation strategies.

1. The system could store internally the graphs and find the shortest path between two nodes on demand by performing a graph traversal. If there are n nodes and e edges in the graph, this approach takes $O(e)$ space and $O(n^2)$ time, using Dijkstra's algorithm [1].

2. Another approach precomputes the distances between the internal nodes and the leaves and stores them using $O(n^2)$ space. Using Floyd's algorithm [1], this can be done in $O(n^3)$ steps. Although somewhat expensive, this is a one time setup cost. Once a distance table has been filled, distances can be retrieved in constant time. In some cases the distances from one node to all leaves will have to be retrieved and sorted. Even this can be done reasonably fast in $O(n \cdot ln(n))$.

If space is not a concern and changes to the graphs are rare, the second alternative seems more attractive. It is interesting, however, to analyze the special case in which the graphs

are balanced trees, as in the examples so far. In this case there is at most one path connecting any two nodes. Using Dijkstra's algorithm [1], one can produce a list of all nodes in ascending order of distance from a source node in time $O(n \cdot ln(n))$. In this case, the first approach is advisable because it is able to handle dynamically changing graphs, at no additional cost. The condition that there be at most one path connecting any two nodes, and that the graph be a balanced tree, are essential for this estimate to hold. For the purposes of this paper we use the second alternative.

4.4 SEP-Shop: The Prototype implementation

We have developed a prototype, *Search and Preference-Based Navigation in Electronic Shopping* (SEP-Shop), in Prolog to demonstrate the concepts in this paper. I SEP-Shop, the distances between the nodes and the leaves are pre-computed and stored as Prolog facts. There is no need to store the distances between internal nodes, because the system will only be looking for leaf nodes. For similar reasons, the distances between leaf nodes, which represent product codes, are not relevant. Thus the only distances that need to be stored are those between internal nodes and product codes. The predicate `dist(Graph, Node, Leaf_node, Distance)` denotes the `Distance` between a `Node` and a `Leaf_node` in graph `Graph`. It is used as follows:

```
dist(clothing,daywear,'jp 200-1000 d',8.3).
dist(clothing,daywear,'jp 350-1000 d',3.25).
```

If there is no path connecting two nodes, the distance separating them is taken to be infinity. We represent the value of infinity with the atom `top` which is greater than any distance in the graph. This concept is enforced by the the clause `dist(_, _, _, top)`, that is placed at the end of the all other `dist` facts. The underscore character _ denotes an uninstantiated variable that matches any value.

Given a graph `G` and a node `N`, our implementation computes the predicate `reachable((G,N),L)`, where `G` is a graph, `N` a node in `G`, and `L` is a list of pairs containing the leaf nodes of `G` and their associated distances to `N`. The list is in ascending order by distance.

To interact with the user, the system captures through a friendly interface, the desired selection consisting of a graph and a node, for example `(clothing, pants)`. This means that although s/he may be unaware of this, the customer will search for pants using the `clothing` graph.[1] The predicate `reachable` is set as a goal, and it returns the ordered list of leaves. Following this, the systems suggests products to the shopper in the order in which they appear in `L`. The predicate `match` implements this process:

```
match(Selection):-      reachable(Selection, List),
                        dialog(List).
```

[1] Other graphs may exist.

```
?-match((clothing,pant)).

leather pant with code # jp 350-1000 d
Would you like to see more ?  y.

leather pant with code # jp 350-1001 d
Would you like to see more ?  y.

corduroy pant with code # jp 279-1730 d
Would you like to see more ?  y.

corduroy pant with code # jp 510-7578 e
Would you like to see more ?  y.

swimsuit with code # jp 200-1000 d
Would you like to see more ?  n.
```

Figure 3: A sample shopping session where a customer is looking for a pair of pants. (The words in italics are entered by the user.)

The `dialog` predicate shows the first item of a list, asks the user if she or he wants to see another one. If the answer is positive, it continues showing the rest of the list. Figure 3 shows a sample session of *SEP-Shop* where a match for *pants* is requested.

5 Multiple Perspectives

When shopping, we generally we take into consideration more than one aspect of the items for sale. We may have in mind certain color and style preferences. We call these various aspects *perspectives*. Lee and Widemeyer's work [9] is on a single perspective; and so is the

extension to expanded graphs of the previous section. In this section we show how certain operations on graphs allow us to deal effectively with search along multiple perspectives.

In principle the idea is quite simple: different graphs portray different perspectives, each determining different distances between nodes. To use more than one perspective during search, i.e. color and season as well as clothing categories, the various graphs are combined into a multi-dimensional graph. Distances between nodes are now computed in this new graph, and items are suggested using the new distance. The operations we describe here for combining various graphs are computationally savvy in that they do not require the re-computation of paths –a computationally expensive process. Instead, the multi-perspective distance is calculated by combining single-perspective distances.

5.1 Perspectives as Dimensions

We explained how a graph imposes a notion of distance (to be interpreted as utility, below, §7) on a collection of abstract categories and product codes. Different graphs will impose different distance notions. If we think of the perspectives as dimensions, we can think of the distance notion imposed by a number of perspectives conjunctively as a multidimensional concept. For example, each of the perspectives of *clothing, color, cost, season* determines a specific dimension. In order to take them all into account, a multidimensional distance should be used. In two dimensional Euclidean space the distance between two points $< x_1, y_1 >$ and $< x_2, y_2 >$ is computed as a function of the distances in the x and the

y-dimensions:

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{\| x_1 - x_2 \|^2 + \| y_1 - y_2 \|^2} \quad (6)$$

Here $\| x_1 - x_2 \|$, which stands for the absolute value of $x_1 - x_2$, is the distance along the x-dimension and $\| y_1 - y_2 \|$ is the distance along the y-dimension. We can rephrase this equation as:

$$d(P_1, P_2) = \sqrt{d_x(P_1, P_2)^2 + d_y(P_1, P_2)^2} \quad (7)$$

where P_1 and P_2 are points in two-dimensional space, and d_x denotes their distance along the first dimension, x, and d_y along the second dimension, y. This shows that the two-dimensional distance is computed using two single dimension distances, the x-distance and the y-distance. In the same spirit, we propose combining distances in various perspectives to compute a multi-perspective distance as follows,

$$d(P_1, P_2) = \sqrt{\sum_{i=1}^{n} d_i(P_1, P_2)^2} \quad (8)$$

where $d_i(P_1, P_2)$ is the distance between points P_1 and P_2 along the i^{th} dimension.

To take an example from the clothing world, consider the *color* perspective represented in Figure 4. The leaves correspond to codes of products that also appear in the *clothing* graph. Although this information is not part of the *color* graph, we have listed the categories of *clothing,* to which the coded objects belong, in order to facilitate our discussion here. The costs of the arcs have been chosen arbitrarily and are not representative of any particular cost assignment function. There is an obvious bias towards darker colors, a marked preference of green over yellow, and of red over blue and brown.

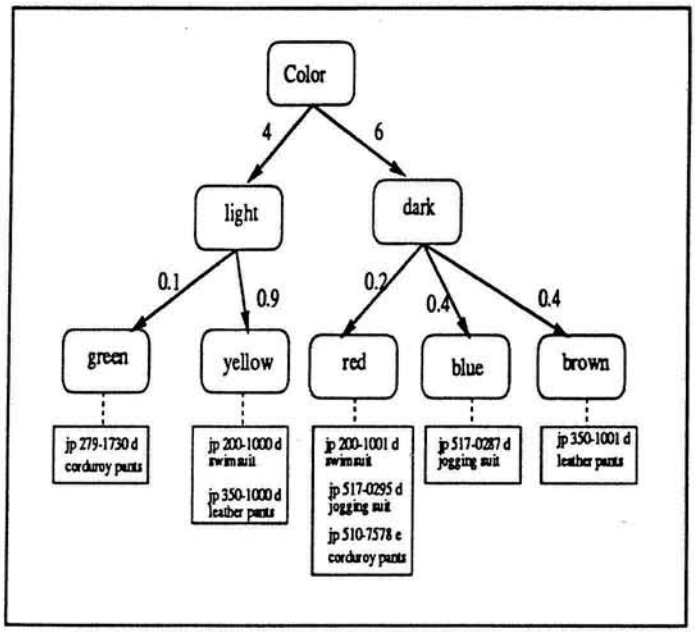

Figure 4: The Color perspective

The *color* graph defines a notion of distance that differs from the one determined by the *clothing* graph. For example, since *jp 279-1730 d*(green corduroy pants) and *jp 510-7578 e* (red corduroy pants) are both corduroy pants, they are very close in the *clothing* perspective (distance= 0.) From the point of view of *color*, however, their distance is 10.3. We would like to combine the information present in both graphs to support a complex search. Suppose the customer is looking for *blue pants*. Notice that there are no blue pants, thus the system has to choose between *brown leather pants* and *red corduroy pants*. From the *clothing* perspective *leather pants* should be considered before *corduroy pants*. From the color perspective *red* should be considered before *brown*. Should it suggest a pair of *brown leather pants* or a pair of *red corduroy pants*? In what follows we will develop tools to help

```
instance_of('jp 279-1730 d',green).              % Color
instance_of('jp 279-1730 d',moderate).           % Cost
instance_of('jp 279-1730 d',female).             % Gender
instance_of('jp 279-1730 d',winter).             % Season
instance_of('jp 279-1730 d','corduroy pants'). % Clothing
```

Figure 5: A sample database entry for the five perspectives of an item.

in these situations.

5.2 A Data Structure for Perspectives

Different perspectives are to be represented by different graphs. As before, the graphs are encoded with the `isa(Graph, Node_1, Node_2, Cost)` predicates. The reason for having a `Graph` argument is that the same nodes might appear in more than one graph (actually all graphs have identical leaves, namely the product codes). The `instance_of` predicate will represents links between the nodes of the graphs and the individual items on stock. A data base containing all the information about the products is to be constructed. If the perspectives taken into account are *clothing, color, cost, gender* and *season*, a typical entry for an item in *SEP-Shop* is shown in Figure 5. By design, the number of perspectives is unconstrained rather than limited to a fixed number of attributes.

5.3 The Multi–Perspective Distance

Elaborating on the analogy between perspectives and dimensions, we propose to compute the multi–perspective distance as a function of the distances on each of the perspectives. One could adopt the *Euclidean* metric:

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \sqrt{\sum_{i=1}^{i=n} d(x_i,y_i)^2} \tag{9}$$

alternatively, one can use a *summation* metric:

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \sum_{i=1}^{i=n} d(x_i,y_i) \tag{10}$$

a *maximal* metric:

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = max_{1\leq i\leq n}(d(x_i,y_i)) \tag{11}$$

or a *minimal* metric:

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = min_{1\leq i\leq n}(d(x_i,y_i)) \tag{12}$$

Each imposes a different penalty on deviation from the optimal point. From a computational point of view it would make sense to find a suitable multi–dimensional distance function that is compositional, i.e. it only depends on the distances in each perspective and does not require finding shortest paths in *multidimensional graphs*. The metrics presented above satisfy this requirement. The distances of the shortest paths in each perspective are combined algebraically to produce the multi-perspective distance. Liberated from the

complexity of finding shortest paths, this method makes it viable to combine -in real time- various perspectives.

The formulæ presented so far provide no means for distinguishing perspectives with regard to relative importance. Under certain circumstances however, there should exist means for specifying that a certain perspective is more important than another one. Suppose, for example, that a customer is looking for a pair of pants, and that he or she would prefer them to be blue. Since, as we saw, there are no blue pants, the system has to find an alternative. In this case it might be better to suggest a brown pair of pants than a blue swimsuit. However, if the color is more important, then the blue swimming suit constitutes a better match. In our implementation we associate weights with the perspectives when performing a search. The distance along each perspective is multiplied by the corresponding weight prior to calculating the multidimensional distance. Thus, following the example, one would assign more weight to *clothing* than to *color*. To do this formally, we use a weighted Euclidean formula.

$$d((x_1,\ldots,x_n),(y_1,\ldots,y_n)) = \sqrt{\sum_{i=1}^{i=n} w_i \cdot d(x_i,y_i)^2} \qquad (13)$$

The weights, w_i, for each perspective should be gathered according to some mechanism that either asks the user for the relative importance of the perspectives, or uses a predetermined scale. Next, we show how to algebraically combine various graphs.

5.4 The Join Operation

The *join* operation combines an arbitrary number of graphs and nodes within those graphs, finding the best match in all perspectives, i.e., the individual that is closest to all of the criterion nodes, where *closest* depends on the metric chosen, which in our case is the weighted Euclidean. As mentioned at the end of the last section, weights are added to the graphs in order to specify their relative importance. The `dist` predicate is extended to compute this distance, it receives an *expanded graph list* (`EGL`) of the form:
$[[Weight_1, (Graph_1, Node_1)], \ldots, [Weight_n, (Graph_n, Node_n)]]$.
The second argument of `dist` is a node `M`. The third argument is the computed distance between the nodes in `EGL` and `M`.

```
dist(join(EGL),M,D):-        coll_dist(EGL,M,L),
                             euclid(L,D).
```

The predicate `coll_dist` is used to collect the distances between a sequence of weight-graph-node triples, `EGL`, and a given node, `M`. We now show the result obtained using the *clothing* and *color* perspectives.

```
?- match(join([[1,(clothing,pants)],[2,(color,blue)]]))
Output:

red corduroy pants with code # jp 510-7578 e
brown leather pants with code # jp 350-1001 d
red swimsuit with code # jp 200-1001 d
blue jogging suit with code # jp 517-0287 d
        .
        .
        .
```

Figure 6: Searching for a blue pair of pants.

Searching for a blue pair of pants.

The search is along two dimensions: `clothing` and `color`. Emphasis is placed on `color` by assigning it twice as much weight as that assigned to `clothing`. Figure 6 is a snapshot form a run of our prototype *SEP-Shop*. Note that although leather pants are preferred over corduroy pants in the clothing perspective, a pair of the latter is suggested first since blue is closer to red than to brown. We envision a friendly user-interface that will elicit customer choices and represent these internally using the join operator. Thus, users of the system would not have to handle the cumbersome notation as it appears in the example.

Searching for a blue pair of pants.

The same query, but now the emphasis is on pants is shown in Figure 7

```
?- match(join([[2,(clothing,pants)],[1,(color,blue)]]))
Output:

brown leather pants with code # jp 350-1001 d
red corduroy pants with code # jp 510-7578 e
green corduroy pants with code # jp 279-1730 d
yellow leather pants with code # jp 350-1000 d
red swimsuit with code # jp 200-1001 d
        .
        .
        .
```

Figure 7: Searching for a blue pair of pants.

Here, *leather pants* appear before *corduroy* ones, but only red ones. Also, swimsuits appear only after all pants have been shown.

5.5 The Union Operation

Suppose someone is interested in buying pants, would like them to be blue, but would accept green. This preference can be handled by searching for green and for blue and selecting the best of both outcomes. This operation is implemented via the `union` operator. As with `join`, it takes as an argument an expanded graph list *(EGL)* whose elements are triples containing a weight, a graph and a node in that graph. The distance from each node appearing in the *EGL* to a target node is computed, multiplied by the weight, then the minimum of the distances is taken to be the distance of the `union`. The role of weights is

reversed here due to the use of minima: a higher weight means less importance.

```
dist(union(EGL), M, D) :-      coll_dist(EGL,M,L),
                               minimal(L,D).
```

The situation above is represented by a combination of `join` and `union` as follows:

```
?- match(join([ [3,  union([ [0.3,(color,blue)],
                             [0.7,(color,green)]])],
                [1,(clothing,pants)]])).
Output:

green corduroy pants with code # jp 279-1730 d
brown leather pants with code # jp 350-1001 d
red corduroy pants with code # jp 510-7578 e
yellow leather pants with code # jp 350-1000 d
```

Since the notation can become quite cumbersome, we remind the reader that, in any commercial-grade application, a front-end user interface should handle friendly dialogs while constructing the formal queries in the background. By analyzing the output of *SEP-Shop* we realize that even although green was given lower priority than blue, a green pair of pants is suggested first, since no blue ones are available. The ordering of the subsequent

suggestions comes from the fact that red and brown are closer to blue than yellow is to green.

The same operation can be used to combine different graphs. The color perspective of Figure 4 clusters colors according to their brightness. One could, however, easily think of other interesting criteria, for example complement. Suppose we have a perspective *complementary colors* where red and green are close together as are lilac and yellow. When searching for a red item, the system uses both the *color* and the *complementary colors* perspectives to make suggestions. If nothing red is available, the system suggests an item that matches red either according to brightness or according to complement by using the `union` operator as follows,

```
?- match(join([  [1, union([  [0.5, (color, red)],
                              [0.5, (complementary, red)]])],
                 [1, (clothing, jacket)]])).
```

6 Shortage

So far we have focused on the *surplus* problem, i.e., how to narrow down on the set of possible suggestions and find the best one. In terms of our graph representation, we start at one or more non-leaf nodes and find a closest leaf. The *shortage* problem occurs when a specific item is selected by the user (given by its code number) that is not available. In

terms of our graph representation, we start at a leaf and need to find a different leaf that is as close as possible. Shortage can be dealt with in much the same way as surplus.

Suppose we are looking for item *T-shirt jp 522-1635 d*, and there are none in stock. The query `match('jp 522-1635')` would not deal the desired results because we are not specifying under what perspectives to search. The solution is to have the system find out to which perspectives *jp 522-1635* belongs to and then perform a join operation on these perspectives .

Why it is necessary to perform a join operation? Suppose that we also have a perspective for the cost of items in dollars, so that we know whether or not an item is expensive. If we are looking for the specific T-shirt *jp 522-1635 d,* which happens to be blue and inexpensive, the alternatives offered should consider not only the fact that we want a T-shirt, but also that it should be blue and not high-priced. A default weight assignment for the perspectives will be used to reflect the relative importance of color, cost, etc.

The implementation is as follows:

```
match(Ind):-    individual(Ind),
                collect_categories(Ind, CatList),
                get_weights(CatList,WCL),
                match(join(WCL)).
```

7 Personal Preferences

In many occasions, the may be a natural need to replace certain portions of a graph, especially in what pertains to costs. If for example, a customer's color preferences are not properly represented by the `color` graph, the customer may be given the option of using his/her own color preferences. In this section we deal with these issues.

The operations on graphs that we have so far described are compositional in that no new graph traversals are needed for their implementation. We now introduce an operation that requires graph traversals, because it modifies the underlying graphs.

Suppose that Susan prefers dark colors to light ones. Otherwise her preferences agree with those represented by the *color* perspective. Instead of building a new perspective for her preferences and thereby duplicating most of the graph, it would be interesting to establish a way of changing the values of some arc costs in the *color* perspective. We propose the following solution. Build a graph that only holds the arcs whose costs have to be changed. When computing distances on this new graph default to the other one when no information is available. That is, if there is an arc between two nodes in the new graph, then use that arc; otherwise, rely on the other graph.

For the example of Susan, we would build a special graph, *susan_colors*, containing only the nodes *color, light* and *dark* as shown in figure 8. This kind of default is implemented by altering the `isa` hierarchy as follows,

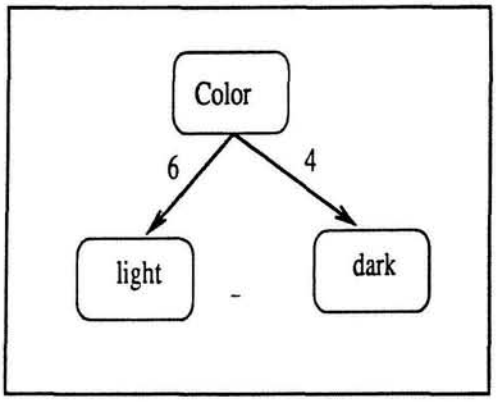

Figure 8: Susan's Preferences

```
isa(default(Graph, Def_Graph), N1, N2, C) :-
                    (isa(Graph, N1, N2, C);
                     isa(Def_Graph, N1, N2, C)).
```

If an arc is present in `Graph` it is used, otherwise one in `Def_Graph` is used.

The advantage of using this *default* operation, as opposed to building a new graph for each minor variance, is twofold. First, it is space efficient since no information is duplicated. Second, it provides for consistency by keeping only one version of the shared information.

By far the most interesting use of defaults is the representation of personal preferences. By interacting with customers and monitoring their choices, the system could observe in which way a customer's preferences differ from the stored ones. It could then build user models in the form of personal perspectives, preferences and weights.

8 Relationship to Utility Theory

Our purpose in this section is to show how our distance measures for a single perspective can be interpreted as a series of unidimensional utility functions, and to show how our extension to multiple perspectives can be interpreted as a series of multiattribute utility functions.

We begin with the unidimensional case, i.e., with a single perspective as in §3. The distance between any two nodes in a single graph, $d(x_i, x_j)$, was (letting $C_{i,j} = C_{j,i}$, for all i, j) simply the length of the path between x_i and x_j. (We have implicitly been assuming that the distance from any node to itself is 0. If we restrict our discussion to trees, there is exactly one path between any two nodes. We relaxed this assumption, and this led us to measure the distance as the length of the shortest path.) Given this, we can readily see that the graph may be taken as encoding a series of conditional utility functions, one for each node. We can define $u_{x_i}(x_j)$, the utility of going to node j given that you are at node i, as

$$u_{x_i}(x_j) = \texttt{max} - d(x_i, x_j) \tag{14}$$

where `max` is the length of the longest path in the graph. Thus, u ranges from 0 to `max`. Since utility functions are unique only up to a positive linear transformation and since in equation 14 the distance is a negative linear transformation of the utility, we minimize distance in order to maximize utility.

Given the definition implicit in 14, it remains to investigate the requirements of that utility function and to determine whether these requirements are reasonable. So far as we are aware, e.g., [4, 7, 8], the sort of graph-functional utility function we are proposing has not been investigated. We will confine ourselves to but a few remarks. We note two properties of our utility function. First, for any node x_k on the shortest path from x_i to x_j,

$$u_{x_i}(x_j) = \max - (d(x_i, x_k) + d(x_k, x_j)) \tag{15}$$

We call this the *additivity* property, since the utility depends on the sum of the arc lengths. Second, we note an *independence* property: $u_{x_i}(x_j)$ is independent of the cost of any arc not on the path from x_i to x_j, the utility is unaffected by changes in the graph not on the path.

These are, we think, sensible properties for a utility function for this sort of application. In any case they can be used diagnostically in eliciting a utility function and constructing a tree. To illustrate, suppose that x_i and x_j are two leaf nodes with a common ancestor, x_k, and that $d(x_i, x_k) < d(x_j, x_k)$. Then, for any node, x_l for which x_k is on the shortest path between x_l and x_i and between x_l and x_j, $u_{x_l}(x_i) > u_{x_l}(x_j)$. This fact can be used to validate a given graph and assignment of arc costs. Further, if upon examination the graph is found to be invalid in this way, then the graph can be modified by adding (or perhaps removing) nodes and arcs, e.g., by splitting x_k so that it is not a common ancestor for both x_i and x_j. Similarly, if the independence property is violated, then the graph can be changed so that the offending arcs are in fact on the paths in question. In any case, changing the underlying graph can be a rather computationally expensive. Further

exploration of these ideas is left for future research.

We turn now to the multiple perspectives case, discussed in §4. The required utility function definition is mainly a generalization of that for the unidimensional case:

$$u_{y_1,\ldots,y_n}(x_j) = \texttt{MAX} - d(x_j, (y_1, \ldots, y_n)) \tag{16}$$

where x_j is a leaf node (hence common to all the trees in question); the y_is are categorization (non-leaf) nodes, one for each perspective in play; and MAX is the length of the longest path in all of the perspectives. In §4 we emphasized that several different distance metric were possible. We choose, as indicated earlier in the discussion of the code, a weighted Euclidean metric for our implementation:

$$d(x_j, (y_1, \ldots, y_n)) = \sqrt{\sum_{i=1}^{i=n} (k_i \cdot w_i \cdot d(x_j, y_i))^2} \tag{17}$$

where, as above, x_j is a leaf node (hence common to all the trees in question); the y_is are categorization (non-leaf) nodes, one for each perspective in play; the w_i are the weights placed on the various perspectives; and the k_i are standardization factors, set so that $k_i \cdot \texttt{max}_i = k_j \cdot \texttt{max}_j = \texttt{MAX}$ for all i, j. (For the sake of simplicity in our implementation, we absorbed the k_is into the w_is.)

This weighted Euclidean metric is, we think sensible and intuitive. However, it can be easily changed for another metric. In particular, we note that with a slightly simpler metric:

$$d(x_j, (y_1, \ldots, y_n)) = \sum_{i=1}^{i=n} (k_i \cdot w_i \cdot d(x_j, y_i)) \tag{18}$$

we have an additive multiattribute utility function, which is the one most commonly used in practice [7].

The fact, that our distance measures for single and multiple perspectives can be viewed as utility functions, is crucial both theoretically and practically. Since utility theory is widely regarded as the best normative theory of rational choice, it is comforting that our representation scheme coheres with that theory. In addition, this gives us, among other things, opportunities to validate the assignment of distances, as well as ways to predict users' preferences. In fact, these two features are two sides of the same coin, as we shall now explain briefly.

Utility theory tells us that preference is (or should be) transitive. If A is preffered to B, and B to C, then A should be preferred to C. We can both predict that users will have transitive preferences and use this fact to validate our preference models. Thus, if under a particular representation intransitive preferences are discovered, this can be treated as indicating a need to revise our model, or representation. In the present context, for example, if there is a path from A to B and from B to C, we would predict that A is preferred to B, B to C, and A to C. If users indicate a contrary preference ordering for A, B, and C, this would tend to show that our graphs were inaccurate and needed revision. On the other hand, if a number of such predictions are extracted from the graphs and confirmed by users, this would tend to increase our confidence that the graphs were indeed adequately representing the users' preferences.

Deeper tests of validity are possible, but we shall confine ourselves to but one more example. Suppose the user wants A, but we are in a shortage situation with both B and C available as alternatives. By using one or more graphs (depending on whether we are in a multidimensional situation), we determine the utilities of B and C, given a preference for A, i.e., we have $u_A(B)$ and $u_A(C)$. At this point there are a very large number of lotteries we can construct in order to validate our graphs (as representations of users' preferences). Suppose, without loss of generality, that $u_A(B) < u_A(C)$. We might, for example, offer a subject a choice between B for certain, or a lottery with a probability p of getting C and a probability $1 - p$ of getting nothing. In such a context, utility theory will help us make predictions. If, for example, $u_A(B) = p * u_A(C)$, then t he subject should be indifferent between the gamble and B for certain, and if $u_A(B) > p * u_A(C)$, then the subject should prefer B to the gamble. In these cases, and many others, it is possible to validate the graph (and its assigned path lengths) by making predications of subjects' preferences, and determining whether these predictions obtain.

The great utility, as it were, of utility theory in this context naturally raises the following questions. Why rely on the graphs at all? Why not, in particular, simply construct a utility function algebraically and use it to evaluate preferences? In principle, this could be done, just as, in principle, books could be printed before the invention of movable type. A great advantage of this graph-based approach is that the graphs can easily be modified with minimal disruption of the utility functions. The graphs effectively encode very many utility functions. In the case of shortage, the graphs encode a utility function for each item

that can be in shortage. This is evident from our notation. How many such functions are there implicit in our scheme? Essentially one for each node in each graph, i.e. for each internal node N in a graph G, there is a utility function $u_{N,G}$. Thus, for example, if the catalog is augmented with a new product, say rubber river rafts, we may, with considerable confidence, add the new product to specific points in the various graphs. We know, for example, that the raft falls under the sporting goods category, rather than the clothing category. This is knowledge that the ordinary algebraic utility models do not exploit, since they do not attempt to capture the abstraction hierarchies expressed by these graphs. Because the method described here does exploit this knowledge, we are able to augment the graphs easily, thereby creating new utility functions more or less as a byproduct of using commonsense judgments.

9 Summary and Conclusion

In this paper, we showed how an abstraction (or `isa`) hierarchy with an imposed distance metric can be used as a representational basis for modeling the salesperson's role (as embodied in the surplus and shortage problems) in an electronic shopping system. Further, we indicated how the distance metric, in the context of the abstraction hierarchy, can be interpreted as a unidimensional utility function. Finally, we extended the single dimensional (single perspective) treatment to multiple dimensions, or *perspectives,* and showed how the resulting representation can be interpreted as a multiattribute utility function,

and we argued that the resulting function is plausible and, most importantly, testable.

If, in the future, there are to be large-scale electronic shopping systems, they will need to accommodate both (a) a large number of products, many of which are close substitutes, and (b) a heterogeneous body of customers who have complex, multidimensional—and perhaps rapidly changing—preferences regarding the products for sale in the system. Further, these systems will have to be designed in a manner so as to both (c) reduce the complexity of the shopping problem from the customer's point of view, and (d) effectively and insightfully match products to customers' needs. We think that our approach, described above, bids fair to be able to meet requirements (c) and (d) in the context of (a) and (b). Of course, no approach can be shown to be optimal. Much remains to be learned, then, about alternative approaches and about refinements to the one we have proposed.

References

[1] A. Aho, J. Hopcroft, and J. Ullman. *Data Structures and Algorithms.* Addison-Wesley Publishing COmpany, 1983.

[2] Robert H. Bonczek, Clyde W. Holsapple, and Andrew B. Winston. *Foundations of Decision Support Systems.* Academic Press, New York, 1981.

[3] Eric K. Clemons and Steven O. Kimbrough. Information Systems, Telecommunications, and Their Effects on Industrial Organization. In Leslie Maggie et al., editor,

Proceedings of the the Seventh International Conference on Information Systems, pages 99–108, December 1986.

[4] Peter C. Fishburn. *Utility Analysis for Decision Making.* Robert E. Kreiger Publishing Company, Huntington, New York, 1979.

[5] Donalds T. Hawkins. Lessons from the Videotex School of Hard Knocks. *ONLINE*, pages 87–89, January 1991.

[6] Donalds T. Hawkins. Videotex Markets, APplications and Systems. *ONLINE*, pages 97–100, March 1991.

[7] Ralph L. Keeney and Howard Raiffa. *Preferences with Multiple Objectives: Preferences and Value Tradeoffs.* John Wiley & Sons, New York, NY, 1976.

[8] David H. Krantz, R. Duncan Luce, Patrick Suppes, , and Amos Tversky. *Foundations of Measurement, Volume I, Additive and Polynomial Prepresentations.* Academic Press, New York, NY, 1971.

[9] Ronald M. Lee and George Widmeyer. Shopping in the Electronic Marketplace. *Journal of Management Information Systems*, 2(4):21–35, 1986.

[10] Michael A. Noll. Videotex: ANatomy of a Failure. *Information and Management*, 9:99–109, 1985.

[11] LINK Resources. United States Consumer Videotex Forecast, 1987-1992. Research Report 206, New York, NY, 1987.

[12] Paul J.H. Schoemaker. The Expected Utility Model: Its Variants, Purposes, Evidence and Limitations. *Journal of Economic Literature*, XX:529–563, June 1982.

[13] Richard Thaler. Toward a Positive Theory of Consumer Choice. *Journal of Economic Behavior and Organization*, 1:39–60, 1980.

[14] Amos Tversky and Daniel Kahneman. The Framing of Decisions and the Psychology of Choice. *Science*, 211:453–458, January 1981.

[15] Oliver E. Williamson. *Markets and Hierarchies.* The Free Press, New York, NY, 1975.

[16] Oliver E. Williamson. The Economics of Organization: The Transaction Cost Approach. *American Journal of Sociology*, 87:548–575, 1981.

Contents

List of Figures

Example 7 The TC_g query **Q**

$$\{<< o, x >, < o', x' >, t > \mid R(o, x, t) \wedge Q(o', x', t)\}$$

is mapped into:

$$[e.A, e'.A' : t]\ (\exists x)(\exists x')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e') \wedge t \in e'.l \wedge e'.A(t) = x'))$$
$$\wedge R(e) \wedge t \in e.l \wedge Q(e') \wedge t' \in e'.l$$

This expression for $\Gamma_{UG}(\mathbf{Q})$ could be simplified (using standard techniques of logical transformation) to

$$[e.A, e'.A' : t]\ R(e) \wedge t \in e.l \wedge Q(e') \wedge t \in e'.l$$

However, this simplification is not always possible as the following example shows.

Example 8 The TC_g query

$$\{< o, x >, t \mid R(o, x, t) \wedge (\exists o')Q(o', x, t)\}$$

is mapped into

$$[e.A : t]\ (\exists x)[R(e) \wedge t \in e.l \wedge e.A(t) = x \wedge (\exists e')(Q(e') \wedge t \in e'.l \wedge e'.A(t) = x)]$$

Note that in this case the variable x serves to equate the terms $e.A(t)$ and $e'.A(t)$, via transitivity. Also note that the quantified group-id variable $(\exists o')$ was replaced with the historic variable $(\exists e')$ in the L_h formula.

Example 9 The TC_g query

$$\{< o, x >, t \mid R(o, x, t) \wedge (\exists x')(\exists t')(R(o, x, t) \wedge Q(o, x', t') \wedge x = x')\}$$

is replaced with

$$[e.A : t]\ (\exists x)((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (\exists x')(\exists t')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge$$
$$(Q(e) \wedge t' \in e.l \wedge e.A(t') = x') \wedge x = x')) \wedge R(e) \wedge t \in e.l$$

This expression can be simplified to

$$[e.A:t]\ (\exists x)(\exists x')(\exists t')((R(e) \wedge t \in e.l \wedge e.A(t) = x) \wedge (Q(e) \wedge t' \in e.l \wedge e.A(t') = x') \wedge x = x') \wedge R(e) \wedge t \in e.l$$

Note that the equality $x = x'$ did not change in the conversion process. However, it follows from the facts that $e.A(t) = x$, $e.A(t') = x'$ and $x = x'$ that the terms $e.A(t)$ and $e.A(t')$ are equal. Also note that the domain variable x' in the TC_g formula remained unchanged in the L_h formula.

Example 10 The TC_g formula

$$\{< o, x >, t \mid R(o, x, t) \wedge \neg Q(o, x, t)\}$$

is converted to

$$[e.A:t]\ (\exists x)(R(e) \wedge t \in e.l \wedge e.A(t) = x \wedge \neg(Q(e) \wedge t \in e.l \wedge e.A(t) = x)) \wedge R(e) \wedge t \in e.l$$

Note that in the previous examples, Γ_{UG} maps safe TC_g formulae into safe L_h formulae. We generalize these observations in the following proposition.

Proposition 4 *Γ_{UG} maps safe TC_g formulae into safe L_h formulae.*

Sketch of Proof: Let ϕ be a safe TC_g formula. We will prove that $\Gamma_{UG}(\phi)$ is safe by verifying all the conditions in the definition of safety for L_h formulae. First, $\Gamma_{UG}(\phi)$ does not have universal quantifiers since ϕ does not have them.

Second, the range expression $\psi_i = (\exists x_{ij_1}) \ldots (\exists x_{ij_i}) R_i(o_i, x_{i1}, \ldots, x_{in_i}, t)$ is mapped into the expression $(\exists x_{ij_1}) \ldots (\exists x_{ij_i})(R_i(e_i) \wedge t \in e_i.l \wedge e_i.A_{ij_1}(t) = x_{i1} \wedge \ldots \wedge e_i.A_{ij_i}(t) = x_{ij_i})$ and also the expression $R_i(e_i) \wedge t \in e_i.l$ is added at the "outermost" level of $\Gamma_{UG}(\phi)$ because of condition 1 in the definition of the mapping Γ_{UG}. Clearly, the two expressions are semantically equivalent. But the second condition was added to make $\Gamma_{UG}(\phi)$ syntactically safe. Since $\Gamma_{UG}(\phi)$ has the formula $R_i(e_i) \wedge t \in e_i.l$ for each range expression at the outermost level, the second condition of safety for L_h formulae is satisfied.

Third, subformula $F_1 \vee F_2$ in ϕ is mapped into $\Gamma_{UG}(F_1) \vee \Gamma_{UG}(F_2)$ so that $\Gamma_{UG}(F_1)$ and $\Gamma_{UG}(F_2)$ have the same set of atoms $t_i \in e_j$ because the formulae F_1 and F_2 have the same set of pairs $< o_j, t_i >$ and because the mapping Γ_{UG} translates them into expressions $t_i \in e_j$.

Finally, the mapping Γ_{UG} is defined so that all the three items in the definition of safety related to maximal conjuncts are satisfied. □

Theorem 5 *$M_{TG} = (TG, L_h)$ is strongly complete with respect to $M_{TU_g} = (TU_g, TC_g)$.*

Sketch of Proof: First of all, Ω_{UG} is clearly a correspondence mapping, and 1-1, because of our grouping axioms. Secondly, the mapping Γ_{UG} satisfies the second condition in the definition of strong completeness for the following reasons. Intuitively, the predicate $R(o, x_1, \ldots, x_n, t)$ in TC_g is mapped into the expression $R(e) \wedge t \in e.l$, so that the historical variable e corresponds to the group-id o and t is in the lifespan of e. Furthermore, group-id's are defined so that the variables $x_1, \ldots, x_n$ are uniquely determined by values of o and t and are irrelevant in the translation process. Also, the expressions $R(o, x_1, \ldots, x_n, t)$ and $R(e) \wedge t \in e.l$ are equivalent. In addition, the mapping Γ_{UG} preserves the structure of the formula ϕ, i.e. it leaves conjunctions, disjunctions and negations of ϕ in their places in $\Gamma_{UG}(\phi)$. □

6.2.3 Mapping L_h Formulae to TC_g

In this section, we define the mapping Γ_{GU} that maps safe L_h formulae into equivalent safe TC_g formulae. Let ϕ be a safe L_h formula. As for the Γ_{UG} mapping, the formula $\Gamma_{GU}(\phi)$ is obtained from ϕ by replacing all the atomic formulae in ϕ together with quantified variables and leaving the structure of ϕ intact (operators $\wedge$, $\vee$, $\neg$ remain unchanged). The replacement of atomic formulae and quantified variables is done in the following manner:

1. Replace quantified variables in L_h as follows.
   (a) Do not change any quantified domain and temporal variables, i.e. $(\exists x)$ and $(\exists t)$ in L_h will remain in $\Gamma_{GU}(\phi)$.
   (b) Replace quantified historic variables $(\exists e_i)$ with $(\exists o_i)$, where o_i is a *unique* group-id variable.
   (c) Consider all pairs of historic and temporal variables e and t such that ϕ contains an expression $t \in e.l$. Depending on the relationship between the scopes of these variables, we add the expression $(\exists x_1) \ldots (\exists x_n)$ to $\Gamma_{GU}(\phi)$, where x_i is a domain variable associated with historic variable e of arity n, as follows.
      i. if t is a free and e is a bound variable, then place the expression $(\exists x_1) \ldots (\exists x_n)$ before the expression $(\exists o)$ obtained in Step 1b;
      ii. if t and e are bound variables, and the scope of e is contained within the scope of t then also place $(\exists x_1) \ldots (\exists x_n)$ before $(\exists o)$;

iii. if t and e are bound and the scope of t is contained within the scope of e then place $(\exists x_1)\ldots(\exists x_n)$ before $(\exists t)$;

iv. in all other cases, do not add anything to the formula.

2. Replace each occurrence of L_h expression $R(e)$ with $(\exists x_1)\ldots(\exists x_n)(\exists t)R(o, x_1, \ldots, x_n, t)$. If e is a bound variable in ϕ, then the group-id variable o is the same as the one that replaced e in the expression $(\exists e)$ in Step 1b. If e is free, then all the occurrences of e are replaced with the same group-id variable o.

3. Replace each occurrence of expression $t \in e.l$ with $R(o, x_1, \ldots, x_n, t)$, where predicate R is *one of* the predicates occurring positively in the maximal conjunct containing $t \in e.l$[14]. If e is a bound variable in ϕ, then the group-id variable o is the same as the one that replaced e in the expression $(\exists e)$ in Step 1, and the domain variables $x_1, \ldots, x_n$ are the same as the quantified variables introduced in Step 1 for the combination of $(\exists e)$ and $(\exists t)$ expressions. If e is a free variable in ϕ, then the group-id variable o and the domain variables $x_1, \ldots, x_n$ are free and are different from all other variables in $\Gamma_{GU}(\phi)$.

4. Replace each term $e.A_i(t)$ in ϕ with x_i, where x_i is defined as follows. Since ϕ is safe, the maximal conjunct containing $e.A_i(t)$ must also contain expressions $t \in e.l$ and $R(e)$ (for some R). In Step 3, $t \in e.l$ is replaced with $R(o, x_1, \ldots, x_n, t)$. Then x_i corresponds to the variable in this expression that corresponds to attribute A_i in R.[15]

Examples illustrating the mapping Γ_{GU} follow. In these examples we assume that the schemas of relations R and Q from L_h are $R(A, B)$ and $Q(A)$ respectively.

Example 11 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge e.B(t) = 5$$

is mapped into the TC_g query as follows. $R(e)$ is replaced with $(\exists x')(\exists y')(\exists t')R(o, x', y', t')$, $t \in e.l$ with $R(o, x, y, t)$, and $e.B(t) = 5$ with $y = 5$.

Putting the pieces together, we get the answer:

$$\{< o, x >, < o, y >, t \mid (\exists x')(\exists y')(\exists t')R(o, x', y', t') \wedge R(o, x, y, t) \wedge y = 5\}$$

[14] It follows from the grouping axioms in Section 6.1 that it does not matter which positively occurring predicate R is selected. Any selected predicate will produce the same results. In fact, all the qualifying predicates can be selected as well, for a longer but logically equivalent formula.

[15] Remark in the footnote 14 is also applicable here.

Since $(\exists x')(\exists y')(\exists t')R(o,x',y',t') \wedge R(o,x,y,t)$ is equivalent to $R(o,x,y,t)$ we can rewrite the previous query as

$$\{< o,x >, < o,y >, t \mid R(o,x,y,t) \wedge y = 5\}$$

Example 12 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge (\exists t')(R(e) \wedge t' \in e.l \wedge (\exists e')(Q(e') \wedge t' \in e'.l \wedge R(e) \wedge t \in e.l \\ \wedge e.B(t) = e'.A(t')))$$

is mapped into the TC_g query

$$\{< o,x >, < o,y >, t \mid R(o,x,y,t) \wedge (\exists x'')(\exists y'')(\exists t')(R(o,x'',y'',t') \wedge \\ (\exists o')(\exists x')(Q(o',x',t') \wedge R(o,x,y,t) \wedge y = x'))\}$$

Note that the domain variable x' in the previous example is quantified in the *same* part of the $\Gamma_{GU}(\phi)$ formula as the group-id variable o'. Also note that the variables x'', y'' are quantified together with temporal variable t'. In general, the domain variables appearing in the same predicate as group-id variable o and temporal variable t are quantified together with the *innermost* scope of variables o and t. The next example shows how Γ_{GU} handles negations.

Example 13 The L_h query

$$[e'.* : t]\ (\exists e)(Q(e) \wedge \neg(t \in e.l \wedge Q(e)) \wedge R(e') \wedge t \in e'.l) \wedge R(e') \wedge t \in e'.l$$

is converted to

$$\{< o',x' >, < o',y' >, t \mid (\exists o)(\exists x)((\exists x'')(\exists t'')Q(o,x'',t'') \wedge \neg Q(o,x,t) \wedge \\ R(o',x',y',t)) \wedge R(o',x',y',t)\}$$

The next example shows that Γ_{GU} does not affect domain variables in ϕ.

Example 14 The L_h query

$$[e.* : t]\ R(e) \wedge t \in e.l \wedge (\exists z)(R(e) \wedge t \in e.l \wedge e.A(t) = z)$$

is translated into

$$\{< o,x >, < o,y >, t \mid R(o,x,y,t) \wedge (\exists z)(R(o,x,y,t) \wedge x = z)\}$$

Proposition 6 *Γ_{GU} maps safe L_h formulae into safe TC_g formulae.*

Sketch of Proof: The proof proceeds along the lines of the proof of Proposition 4. □

Theorem 7 *$M_{TU_g} = (TU_g, TC_g)$ is strongly complete with respect to $M_{TG} = (TG, L_h)$.*

Sketch of Proof: First of all, Ω_{GU} is clearly a correspondence mapping, and 1-1, because of our grouping axioms. Secondly, the mapping Γ_{GU} satisfies the second condition in the definition of strong completeness, as we shall show by by induction on maximal conjuncts in ϕ in L_h. At any inductive step the L_h formula $\phi(e_1, \ldots, e_n, x_1, \ldots, x_m, t_1, \ldots, t_k)$ is mapped into the TC_g formula $\Gamma_{GU}(\phi)(o_1, \ldots, o_n, x_1, \ldots, x_m, y_1, \ldots, y_l, t_1, \ldots, t_k)$, where $y_1, \ldots, y_l$ are extra variables introduced in the translation process (i.e. when $R(e) \wedge t \in e.l$ becomes $R(o, y_1, \ldots, y_s, t)$). Notice that variables $y_1, \ldots, y_l$ are uniquely determined (i.e. functionally depend) by values of variables $o_1, \ldots, o_n, x_1, \ldots, x_m, t_1, \ldots, t_k$. Therefore, these variables are "superfluous" and do not affect the translation process. With this observation in mind, the proof proceeds along the lines of Theorem 5. □

The following theorem immediately follows from Theorem 5 and Theorem 7.

Theorem 8 *The grouped model $M_{TG} = (TG, L_h)$ and the ungrouped model with group identifiers $M_{TU_g} = (TU_g, TC_g)$ are strongly equivalent.*

Theorems 3 and 8 establish the connections between grouped and ungrouped historical data models. The power of temporal grouping which is inherent in grouped models can only be achieved in an ungrouped model by the addition of some mechanism, analogous to our group identifiers, for simulating the grouping.

7 Historical Models and Completeness

All of the historical relational data models and languages that have been proposed differ from one another in the set of query operators that they provide. In addition, they often differ in the structure of the historical relations that they specify, that is, the way in which the temporal component is incorporated into the structure. Space obviously precludes an analysis of all of these models with respect to our two notions of completeness. Since we

have two orthogonal characteristics to describe these models and their languages – grouped or ungrouped, algebra or calculus – we decided to discuss four models, each covering one of the four possibilities. Two of the data models we discuss are ungrouped, one with an algebra ([Lor87]) and the other with a calculus ([Sno87]); we therefore investigate whether or not they are *TU-Complete.* The other two data models discussed are grouped, one with an algebra ([CC87]), the other with both an algebra and a calculus ([Gad88]), and so we investigate whether or not they are *TG-Complete.*

We have earlier motivated our choice of L_h and TC as appropriate languages to use for our notions of completeness. Therefore, in this section a data model will be said to be **complete** with respect to $M_{TG} = (TG, L_h)$ (or $M_{TU} = (TU, TC)$) if it is strongly complete with respect to $M_{TG} = (TG, L_h)$ (or $M_{TU} = (TU, TC)$). Although by our definitions we should, strictly speaking, refer to completeness with respect to the data models, we will generally speak more specifically about their languages and apply the term loosely to them. For each of the historical query languages discussed in the following, therefore, we consider first its completeness with respect to either of L_h and TC and vice versa. We shall see that L_h and TC are complete with respect to *all* of the languages we consider, a fact which lends further support to their use as the standards for *TG-Completeness* and *TU-Completeness.*

We begin with a discussion of the completeness of the historical relational algebra specified by the historical relational data model **HRDM** [CC87]. We discuss this language first both because the TG model defined in Section 2 is derived directly from the structure of the historical relations in **HRDM**, and because the set of operators specified by this model were intended initially to provide all the functionality thought useful and desirable.

7.1 HRDM

The historical relational data model **HRDM** presented in [CC87] is a temporally grouped historical data model with an algebraic query language which is presented as an extension to the standard relational algebra.

We can categorize the operators of **HRDM** as follows:

Set-Theoretic These operators are defined in terms of the set characteristics of relations, and include the standard set operators union ($\cup$), intersection ($\cap$), set difference (-), and Cartesian product ($\times$). Because these operators do not exploit the *historical* aspects of

HRDM relations, the standard mappings from these operators in relational algebra to their counterpart in relational calculus also applies to these operators here. For example,

$$\begin{aligned} r \cup s &= \{x | x \in r \vee x \in s\} \\ &\equiv [e. * | t] r(e) \wedge t \in e.l \vee s(e) \wedge t \in e.l \end{aligned}$$

Attribute-Based This category includes those operators that are defined in terms of the attributes (or their values) of a relation. Some of these operators, as suggested by their names, are derived from similar operators that exist in the standard relational algebra. As shown below, often the original definition of these operators has been modified to exploit the temporal component of the historical model. For each of these operators we give both its set-theoretic definition, and then an equivalent L_h-based expression.

1. **Project (π):** This operator is equivalent in definition to its standard relational counterpart, and has the effect of reducing the set of attributes over which each of the tuples x in its operand, a relation r, is defined, to those attributes contained in a set of attributes X.

$$\begin{aligned} \pi_X(r) &= \{x(X) | x \in r\} \\ &\equiv [e.X : t] r(e) \wedge t \in e.l \end{aligned}$$

2. **Select-If (σ-IF):** This variant of the select operator selects from a relation r those tuples x each of which for some period within its lifespan has a value for a specified attribute A that satisfies a specified selection criterion. The period of time within the lifespan is specified by a lifespan parameter L. The selection criterion is specified as $A\theta a$, where θ is a comparator and a is a constant. (It is also possible to compare one attribute with another in the same tuple.) A parameter, Q, of the select-if operator is used to denote a quantifier that specifies whether the selection criterion must be satisfied for all ($\forall$) times in the specified subset of the tuple's lifespan, or that there exists ($\exists$) at least one such time.

$$\begin{aligned} \sigma\text{-IF}_{(A\theta a, Q, L)}(r) &= \{x \in r | Q(t \in (L \cap x.l))[x.A(t)\theta a]\} \\ (\text{if } Q \text{ is } \exists) &\equiv [e. * : t] r(e) \wedge t \in e.l \wedge \\ &\quad \exists t_1 (t_1 \in L \wedge t_1 \in e.l \wedge e.A(t_1)\theta a) \\ (\text{if } Q \text{ is } \forall) &\equiv [e. * : t] r(e) \wedge t \in e.l \wedge \\ &\quad \neg \exists t_1 (t_1 \in L \wedge t_1 \in e.l \wedge \neg e.A(t_1)\theta a) \end{aligned}$$

3. **Select-When (σ-$WHEN$):** This operator is similar to the $\exists$-quantified select-if operator. However, the lifespan of each selected tuple is restricted to those times *when* the selection criterion is satisfied[16].

$$\begin{aligned}\sigma\text{-}WHEN_{A\theta a}(r) &= \{x|\exists x' \in r[x.l = \{t|x'.A(t)\theta a\} \wedge x.v = x'.v|_{x.l}]\} \\ &\equiv [e.* : t]r(e) \wedge t \in e.l \wedge e.A(t)\theta a\end{aligned}$$

4. **θ-Join:** Like its counterpart in the standard relational data model this operator combines tuples from its two operand relations. With θ-join two tuples are combined when two attributes, one from each tuple, have values at some time in the intersection of the tuples' lifespans that stand in a θ relationship with each other. The lifespan of the resulting tuple is exactly those times when this relationship is satisfied.

   Let r_1 and r_2 be relations on schemes R_1 and R_2, respectively, where $A \in R_1$ and $B \in R_2$ are attributes.

$$\begin{aligned}r_1[A\theta B]r_2 &= \{e|\exists e_{r_1} \in r_1, \exists e_{r_2} \in r_2 e.l = \{t|e_{r_1}(A)(t)\theta e_{r_2}(B)(t)\} \wedge \\ &\quad e.v(R1) = e_{r_1}.v(R1)|_{e.l} \wedge \\ &\quad e.v(R2) = e_{r_2}.v(R2)|_{e.l}]\} \\ &\equiv [e_1.*, e_2.* : t]r_1(e_1) \wedge r_2(e_2) \\ &\quad \wedge t \in e_1.l \wedge t \in e_2.l \wedge e_1.A(t)\theta e_2.B(t)\end{aligned}$$

5. **Static Time-Slice ($\mathcal{T}_{@L}$):** This operator reduces an historical relation in the temporal dimension by restricting the lifespan of each tuple e of the operand relation r to those times in the set of times L.

$$\begin{aligned}\mathcal{T}_{@L}(r) &= \{e|\exists e' \in r[l = L \cap e'.l \wedge e.l = l \wedge e.v = e'.v|_l]\} \\ &\equiv [e.* : t]r(e) \wedge t \in e.l \wedge t \in L\end{aligned}$$

Other Operators

In addition to the above categories of operators, the **HRDM** algebra includes several *grouping* operators that are used to restructure a relation without changing the information content of that relation. These operators, **union-merge** ($\cup_o$), **intersection-merge**

[16]The notation $f|_l$ in this definition, used in **HRDM**, is the standard notation for denoting the restriction of the domain of the function f to the set l.

($\cap_o$), and **difference-merge** ($-_o$), first computes the set-theoretic union, intersection, and difference, respectively, and then regroups the tuples in the resulting relation.

The **HRDM** algebra also includes the operators **WHEN** and **Dynamic Time-Slice**. We categorize the **WHEN** operator as an *extra-relational* operator in that it computes a result that is not contained in a database relation, nor given as a constant. Applied to an historical relation, this operator returns a value defined as the union of the lifespans of the tuples in that relation. This operator can be viewed as a type of temporal-based *aggregate* operator. The dynamic time-slice is only applicable to relations that include in their scheme an attribute A whose domain consists of partial functions from the set $TIMES$ into itself. We do not treat such attributes in this paper since most of the models considered distinguish between ordinary values and the times at which they hold, and do not allow comparisons between them. Therefore it would be unfair to include such an operator in our comparison. We omit the other operators from our discussion of completeness of **HRDM** and the remaining languages that we will examine. The grouping operators are not treated because they are not intended for querying, and the aggregate operators, because they are outside of the scope of standard relational-based notions of completeness.

The translations that we have provided for each of the relation-defining operators of the **HRDM** algebra shows that L_h is complete with respect to this algebra. However, this algebra is *not TG-Complete* in that there are queries that are expressible in L_h for which no equivalent algebraic expression (i.e., sequence of algebraic operations) exists. One example is the query on the database in Figure 8 for the name and department of each employee that has at some time received a cut in salary, expressible in L_h as

$$[e.NAME, e.DEPT : t]\mathbf{EMPLOYEE}(e) \wedge t \in e.l \wedge$$
$$\exists t_1 \exists t_2(\mathbf{EMPLOYEE}(e) \wedge t_1 \in e.l \wedge t_2 \in \wedge e.l \wedge$$
$$(t_1 < t_2) \wedge e.SAL(t_1) > e.SAL(t_2))$$

The lack of an equivalent algebraic expression is due to the specification of those operators in **HRDM** that include the comparison of two values as part of their definition: the join, and the various select operators. In each case only attribute values that occur at the same point in time can be compared. (This ability seems to be what is meant by the property of supporting "a 3-D conceptual view of an historical relation" that has been cited as an intuitively necessary component of a good temporal database model (e.g. in [CT85, Ari86, MS91a].) Thus, as required by the above query, it is not possible to compare the salary of

an employee at some time t_1 with that employee's salary at some other point in time, t_2.

7.2 The Historical Homogeneous Model of Gadia

The next historical model that we discuss is one that was proposed by Gadia [Gad88]; it is a model that includes a query language and an algebra. This data model, which we shall label **TDMG**(for Temporal Data Model of Gadia), is the same as that of **HRDM**, and thus of the canonical historical model *TG* defined in Section 2.

In **TDMG** the value of a tuple attribute is a function from a set of times to the value domain of the attribute, and the lifespan is the same for all the attributes (Gadia's *homogeneity assumption*). Therefore the **TDMG** model is *temporally grouped.*

In addition to the data model, Gadia defines an historical algebra and calculus. Although his data model is temporally grouped, the semantics of the algebra is defined in terms of the ungrouped model obtained by ungrouping temporal relations. Gadia calls this a *snapshot interpretation* semantics. The semantics of the historical algebra is defined by ungrouping temporal relations because Gadia considers grouped and ungrouped models "weakly equal" and does not distinguish between them when he proves equivalence of his algebra and calculus. In terms of our discussion on completeness in Section 3, Gadia's mapping from his grouped model to his ungrouped model is not a 1-1 mapping; unlike our mapping into TC_g, Gadia's Ω mapping ignores grouping.

Gadia's (ungrouped) algebra is defined as follows. He starts with the five standard relational operators, selection, projection, difference, Cartesian product, and union, as *TA* does. He also defines derived temporal operators such as join, intersection, negation, and renaming. In addition, he defines temporal expressions for the temporal domain. Finally, he combines relational and temporal expressions by considering relational expressions of the form $e(v)$ where e and v are relational and temporal expressions, respectively.

TC is complete with respect to Gadia's algebra for the following reasons. The five standard temporal operators are defined as for *TA* and, therefore, can be expressed in *TC*. Temporal expressions are defined as a closure of a time intervals over the operations of union, intersection, difference and negation. Each of these operators can be expressed in the first-order logic with explicit references to time. For example, the expression $tdom(r(A,B)) \vee tdom(s(A,B))$ in **TDMG** can be defined in *TC* as $\{t \mid (\exists x)(\exists y)(r(x,y,t) \vee s(x,y,t))\}$. This

means that every query in **TDMG** can be expressed in *TC*.

Gadia also defines an historical calculus and shows its equivalence to the algebra (modulo temporal grouping). This calculus is expressible in L_h for the same reasons that the ungrouped algebra is expressible in *TC*. A lifespan of a temporal tuple x in **TDMG** can be captured with expression $t \in x.l$ in L_h. Also, the operators of union, intersection, difference and negation for temporal expressions can be expressed in L_h with the same methods that are used to express algebraic expressions in *TC* since L_h explicitly supports time.

The temporally grouped language L_h has strictly more expressive power than Gadia's calculus, i.e. this calculus is not *TG-Complete*. Also, the temporally ungrouped language *TC* is strictly more powerful than Gadia's algebra, i.e. the algebra is not *TU-Complete*. The reason for this lack of completeness is the same as for **HRDM**: it is not possible to compare the value of one attribute at time t_1 with the value of another or the same attribute at some other time t_2. For example, the query of the previous section, asking for the name and department of each employee that has at some time received a cut in salary, cannot be expressed in **TDMG**.

7.3 TQuel

TQuel is the query language component of an historical relational data model proposed by Snodgrass [Sno87]. We shall call this model **TRDM**.

TRDM provides for two types of historical relations. One, called an **interval** relation, is derived from a standard relation through the addition of two temporal attributes, *valid-from* and *valid-to*, both of whose domains are the set of times T. (An example of such a relation has already been given in Figure 3). As before, we will ignore the two *TRANS-TIME* temporal attributes since we are only considering *historical* data models. Thus we will view **TRDM** as a temporally ungrouped historical data model. The values of the non-temporal attributes of a tuple in such a relation are considered to be valid during the beginning of the interval of time starting at the *valid-from* value and ending at, but not including, the *valid-to* value. (This interval thus denotes the *lifespan* of the tuple.)

The second type of relation, an **event** relation, is defined by extending a standard relation by a single temporal attribute *valid-at*. Since both interval relations and event relations are derived from first normal form relations through the addition of attributes whose values are

atomic, they are also in first normal form.

The query language TQuel is an extended relational calculus derived from and defined as a superset of Quel, the query language of the Ingres relational database management system [SWKH76]. TQuel extends Quel by adding temporal-based clauses that accommodate the *valid-from* and *valid-to* attributes. (These attributes are not visible to the existing components of the Quel language.)

A *WHEN* clause is added to define an additional temporal-based selection constraint that must be satisfied in conjunction with the constraint defined by the TQuel (and Quel) *WHERE* clause. This constraint, specified as a temporal predicate over a set of tuple *valid-from-valid-to* intervals (lifespans) defines a restricted set of relationships that must hold among them. A *VALID* clause is used to define, in terms of temporal expressions, *valid-from* and *valid-to* values for tuples in the relation resulting.from the TQuel statement.

Both temporal predicates and temporal expressions have a semantics that is expressible in terms of the standard tuple calculus ([Sno87]).[17] TQuel is complete with respect to TC, and vice versa, since the semantics of TQuel like that of Quel [Ull88] can be expressed in terms of the standard relational calculus, with which TC is clearly strongly equivalent. In particular, Snodgrass shows how any TQuel query can be expressed as a formula of the form $Q \wedge \Gamma \wedge \Phi$ where Q, Γ, and Φ are the calculus formulae of the underlying Quel statement, the TQuel *WHEN* clause and *VALID* clause, respectively, and Γ and Φ contain no quantifiers. Additionally, Γ and Φ are defined only over the temporal attributes *valid-from* and *valid-to*, neither of which may be included in Q. The structure of this formula means that, as with Quel, not all algebraic expressions can be expressed as a single TQuel statement (for example, algebraic expressions containing the union operator).

If none of the non-temporal attributes over which a **TRDM** database is defined has a domain whose values are comparable to those in the set of times T, then in no algebraic expression over the relations in this database can such an attribute be compared to either *valid-from* or *valid-to.* For such a database, TQuel statements, as represented by a defining tuple calculus formula, are no more restrictive than Quel statements. Therefore (as with Quel) a sequence of TQuel statements can express any algebraic expression, perhaps by creating temporary relations, and using statements such as *APPEND* and *DELETE*,

[17] This specification also includes the use of several auxiliary functions that are used to compare times in order to determine which of two times occurs first or last.

Although interval relations and event relations are distinguished by TQuel, they are standard first normal form relations that provide a fixed way of encoding temporal data using the temporal attributes. TQuel differs from Quel only in the distinction accorded these attributes. Thus, like Quel – with the addition of such statements as *APPEND* – it is complete in the sense defined by Codd. By extension, as a result of the use of the temporal attributes, it is *TU-Complete*, but, like all ungrouped models, it does not exhibit *temporal value integrity.*

We note that the query on the database in Figure 8 for the name and department of each employee that has at some time received a cut in salary, expressible in L_h as

$$[e.NAME, e.DEPT : t]\mathbf{EMPLOYEE}(e) \wedge t \in e.l \wedge$$
$$\exists t_1 \exists t_2(\mathbf{EMPLOYEE}(e) \wedge t_1 \in e.l \wedge t_2 \in \wedge e.l \wedge$$
$$(t_1 < t_2) \wedge e.SAL(t_1) > e.SAL(t_2))$$

is also expressible (again, ignoring transaction times) in **TRDM** as:

```
range of e1 is EMPLOYEE
range of e2 is EMPLOYEE
retrieve into SalChange(e1.NAME, e1.DEPT)
valid from begin of e1 to end of e1
where e1.NAME = e2.NAME AND e2.SAL < e1.SAL
when (end of e1) precede (begin of e2)
```

We note further that an algebra has been proposed that provides a procedural equivalent to the **TRDM** calculus ([MS91b]). While it employs a different data model from that in **TRDM** (in fact, its model is N1NF), it is *not* a grouped model and does not support grouping.

7.4 The Temporal Relational Algebra of Lorentzos

The final historical data model that we discuss is one that was proposed by Lorentzos in [Lor87]. The data model in [Lor87], which is called **TRA**, is essentially the same as that in [Sno87], except that as an *historical* model it is restricted to only one temporal dimension. Two of the stated goals of **TRA** are that "no new elementary relational algebra operations are introduced and first normal form is maintained" [Lor87, p. 99]. Typical relations in this model appear basically as in Figure 3 (with the columns *valid-from* and *valid-to* called *Sfrom*

and *Sto*, respectively). Although the structures of relations in this model are essentially the same as in the historical version of **TRDM**, we discuss this model here because, unlike [Sno87], the language it proposes is an algebra rather than a calculus.

It is difficult to discuss formally the algebra of **TRA** because it is not specified formally. Rather, it is presented via a series of example queries and discussion. Nevertheless, enough of a picture of the algebra emerges clearly through these examples to make a discussion possible.

Two new operators, *FOLD* and *UNFOLD* are defined. These operators essentially convert between the time interval representation (as in Figure 3) and a time point representation (as in Figure 1). The *FOLD* and *UNFOLD* are clearly expressible in terms of operators in the standard relational algebra, as [Lor87] points out.

The previous sections demonstrated that two other algebras, that of **HRDM** and that of **TDMG** were incomplete because they were not able to compare the value of one attribute at a time t_1 with the value of another (or the same) attribute at some other time t_2. In **TRA** such comparisons *are* possible. Consider again the query that finds the name and department of each employee that has at some time received a cut in salary:

$$[e.NAME, e.DEPT : t]\mathbf{EMPLOYEE}(e) \wedge t \in e.l \wedge$$
$$\exists t_1 \exists t_2(\mathbf{EMPLOYEE}(e) \wedge t_1 < t_2 \wedge e.SAL(t_1) > e.SAL(t_2))$$

This query can be expressed in **TRA** as follows. First UNFOLD the interval relation EMPLOYEE into all of its time points:

$$EMPLOYEE_{U_1} = UNFOLD[Time, Start, Stop](TIME, EMPL)$$

Then, Θ-Join this relation with itself, joining tuples with the same name and with a pay cut, and then Project just the names of the employees from the result (here NAME1 and NAME2, etc., refer to the NAME attributes in the first and second operands to the Join):

$$TEMP1 = EMPLOYEE_{U_1} \left[\begin{array}{lcl} NAME1 & = & NAME2, \\ TIME1 & < & TIME2, \\ SAL1 & > & SAL2 \end{array} \right] EMPLOYEE_{U_2}$$

$$TEMP2 = \pi_{NAME1}(TEMP1)$$

Finally, Join the result with the original relation and Project onto the desired fields:

$$\pi_{\{NAME,DEPT,Sfro,Sto\}}\ (TEMP2 \bowtie EMPLOYEE)$$

Language	Reference	Type	Completeness
L_h	Section 5	grouped	Basis for *TG-Completeness*
TC	Section 4	ungrouped	Basis for *TU-Completeness*
TRA algebra	[Lor87]	ungrouped	*TU-Complete*
TRDM calculus	[Sno87]	ungrouped	*TU-Complete*
HRDM algebra	[CC87]	grouped	*not TG-Complete*
TDMG calculus	[Gad88]	grouped	*not TG-Complete*
TDMG algebra	[Gad88]	ungrouped	*not TU-Complete*

Figure 15: Summary of Completeness Results

Because **TRA** is equivalent to standard relational algebra, the question of its *TU-Completeness*, as in the case of **TRDM**, is reduced to the question of the completeness of relational algebra. Therefore we conclude that **TRA** is *TU-Complete* but, like all ungrouped languages, it does not exhibit *temporal value integrity.*

The results of our explorations into the completeness of these five languages is summarized in the Table in Figure 15.

8 Summary and Conclusions

In this paper we have explored the question of completeness of languages for historical database models. In this exploration we were led to characterize such models as being of one of two different types, either **temporally grouped** or **temporally ungrouped**. We first discussed these notions informally by means of example databases and queries, and showed that the two models were not equivalent. The difference between the two models is that in temporally grouped models, historical values (like salary histories) are treated as first class objects which can be referred to directly in the query language. In the temporally ungrouped models, no such direct reference is permitted. We characterized this property of the grouped models as **temporal value integrity**.

We then proceeded to define the two concepts of *weak completeness* and *strong completeness* between two data models with different representation paradigms and different query languages. In the case of weak completeness, there is a correspondence mapping from the relations of the reference model to the comparison model, and a mapping on the query language which preserves the meaning of a query. The problem with weak equivalence is that

different relations in the reference model can be mapped to the same relation in the comparison model, and so information, e.g. grouping, can be lost. In the case of strong completeness, the correspondence mapping must be 1-1, and hence there is no loss of information.

For the ungrouped models we defined three different languages, *TL*, *TC*, and *TA*: a temporal logic, a logic with explicit reference to time, and a temporal algebra, and showed that under certain assumptions about the model of time employed all three are equivalent in power. Any one of the three can serve as the basis for *TU-Completeness.* An ungrouped model is said to be *TU-Complete* if it is strongly complete with respect to $M_{TU} = (TU, TC)$.

For the grouped models we defined the calculus L_h, a many-sorted logic with variables over ordinary values, historical values, and times. We proposed L_h as the basis for *TG-Completeness.* A grouped model is said to be *TG-Complete* if it is strongly complete with respect to $M_{TG} = (TG, L_h)$.

We then proceeded to explore more formally the relationship between ungrouped and grouped models. We demonstrated a technique for extending the ungrouped model with a grouping mechanism, a *group identifier.* With this mechanism we showed how the ungrouped model TU and the language TC could be extended to TU_g and TC_g in such a way as to make the resulting model equivalent in power to TG with L_h. In this way we demonstrated that the grouped and ungrouped models differ only with respect to the *grouping* capability. More precisely, we proved that the model $M_{TU} = (TU, TC)$ is weakly equivalent, and the model $M_{TU_g} = (TU_g, TC_g)$ is strongly equivalent, to the model $M_{TG} = (TG, L_h)$.

Finally, we examined several historical relational proposals to see whether they were *TU-Complete* or *TG-Complete.* We looked at four historical models, two grouped and two ungrouped, offering five different languages. In the ungrouped models we found both an algebra (from **TRA**) and a calculus (TQuel from **TRDM**) which are *TU-Complete*, while in the grouped models we found, apart from our metric, the complete calculus L_h, two languages which are *not TG-Complete*: an algebra (from **HRDM**) and a calculus (from **TDMG**), as well as an algebra (from **TDMG**) (which operates on ungrouped versions of grouped relations) which is not *TU-Complete.* We believe that this classification scheme, and our examination of the completeness of several historical models, should help to explicate the differences and the commonalities between the various models proposed in the literature. As with the relational model, a baseline notion of *completeness* of query languages, while imperfect (e.g. relationally complete languages do not allow for transitive closure queries or support aggregates), nonetheless provides a minimum and reasonable metric with which to

compare a variety of different languages.

One point bears emphasizing. It has on occasion been said that the issue of adding time to relational databases is an uninteresting one, since the user can always just add whatever extra attributes are desired (e.g., **Start-Time** and **End-Time**) and then use standard SQL (or relational algebra) as the query language. In our discussion of the completeness of the ungrouped temporal languages we, to some extent, relied on the underlying point of this argument. For example, this point underlay our argument that **TRA** (which is equivalent to standard relational algebra) is *TU-Complete*. Two points need to be made in reply to this comment. First, there is a difference between the formal notion of completeness and the informal, but no less important, notion of ease of use. Even though the programming language *C* is formally equivalent to a Turing Machine, it is a lot more convenient to use *C* if you are writing an operating system because of its *built-in* high level features. The *built-in* temporal features of the historical and temporal data models make them easier to use for managing temporal data; without these features a greater burden is placed upon the user. Secondly, this paper has shown that the grouped models and languages are more expressive than their corresponding ungrouped models, unless these models add a surrogate grouping mechanism. This grouping mechanism, itself, is a higher-level construct that is *implicit* in the grouped systems (and this, we argue, makes them more convenient), but needs to be made *explicit* in the ungrouped systems for them to be equivalent in expressive power.

There are a few interesting areas for future research that this work has clarified. The first question relates to our grouping axioms (in Section 6). It might seem that they are rather strong, perhaps stronger than necessary for simulating temporal grouping in a temporally ungrouped model like *TU*. Clearly, in order to have an isomorphism between two such models, the Ω structural mapping and the Γ mapping on queries must work hand in hand. It is an area for additional research whether our Ω_{GU} could be simplified, most likely at the expense of complicating the mapping on queries.

Another area of interest arises when it is noted that we did not find here, nor are we aware of, *any* complete algebra for grouped historical data models. Such an algebra is clearly needed. Another area in which there continues to be interest is in the support of evolving schemas. Our decision not to treat this interesting area here was based largely on the fact that hardly any of the models in the literature incorporate this feature, and we wanted to choose the common denominator of all the models in order to make our comparisons fairly. The model in [CC87] addressed this issue, and other work (e.g. [BKKK87, MS90]) continues

to be done in this area.

Finally, we would like to address the question of completeness for *temporal* as opposed to *historical* relational models (in the terminology of [SA85]). We believe that our results on grouped and ungrouped historical relational completeness can be extended in a straightforward way to temporal data models and languages. The extension would involve the addition of another sort (for transaction times). In ungrouped temporal models, relations would be extended with an additional column to stamp every tuple with its transaction time, and the language would have constants, as well as variables, and quantification for this sort. In grouped temporal models, values would be extended to be doubly indexed; they would most likely be better modeled as functions from a transaction time into functions from a data time to a scalar value, but the order of the two temporal indices could be reversed. Preliminary work that we have done on Indexical Databases [Cli92] holds promise for a unified treatment, not only of these two temporal dimensions, but of spatial, or other, dimensions as well.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable comments which have helped to improve the contents and presentation of this paper. We would also like to thank Rick Snodgrass for ongoing and fruitful discussions that have helped to clarify many of the ideas presented here.

References

[AC86] G. Ariav and J. Clifford. Temporal data management: Models and systems. In G. Ariav and J. Clifford, editors, *New Directions for Database Systems*, pages 168–185. Ablex Publishing Corporation, 1986.

[Ari86] G. Ariav. A temporally oriented data model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[AU79] A.V. Aho and J.D. Ullman. Universality of data retrieval languages. In *ACM Symposium on Principles of Programming Languages*, pages 110–120, New York, 1979. ACM.

[Ban78] F. Bancilhon. On the completeness of query languages for relational databases. In *Proc. Seventh Symposium on Mathematical Foundations of Computing*, pages 112–123. Springer-Verlag, 1978.

[BKKK87] J. Banerjee, W. Kim, H.-J. Kim, and H.F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proceedings of ACM SIGMOD Conference*, pages 311–322, San Francisco, CA, 1987. ACM.

[BZ82] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, University of California at Los Angeles, 1982.

[CC87] J. Clifford and A. Croker. The historical relational data model HRDM and algebra based on lifespans. In *Proc. Third International Conference on Data Engineering*, pages 528–537, Los Angeles, February 1987. IEEE.

[CH80] A.K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, October 1980.

[Cli82] J. Clifford. A model for historical databases. In *Proceedings of Workshop on Logical Bases for Data Bases*, Toulouse, France, December 1982.

[Cli92] J. Clifford. Indexical databases. In *Proceedings of Workshop on Current Issues in Database Systems*, Newark, N.J., October 1992. Rutgers University.

[Cod72] E.F. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Data Base Systems.* Prentice-Hall, 1972.

[CT85] J. Clifford and A.U. Tansel. On an algebra for historical relational databases: Two views. In S. Navathe, editor, *Proceedings of ACM SIGMOD Conference*, pages 247–265, Austin, TX, May 1985. acm.

[CW83] J. Clifford and D. S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 6(2):214–254, June 1983.

[Dat83] C.J. Date. *An Introduction to Database Systems, vol. II.* Addison-Wesley, 1983. Reading, MA.

[End72] H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972. New York.

[FVG85] P.C. Fischer and D. Van Gucht. Determining when a structure is a nested relation. In *International Conference on Very Large Databases*, pages 171–180, 1985.

[Gab89] D. Gabbay. The declarative past and imperative future: Executable temporal logic for interactive systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proceedings of Colloquium on Temporal Logic in Specification*, pages 402–450. Springer-Verlag, 1989. LNCS 398.

[Gad86] S.K. Gadia. Toward a mulithomogeneous model for a temporal database. In *Proc. Second International Conference on Data Engineering*, Los Angeles, California, February 1986. IEEE.

[Gad88] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *TODS*, 13(4):418–448, 1988.

[Hal60] P. Halmos. *Naive Set Theory.* D. Van Nostrand, Princeton, NJ, 1960.

[HOT76] P. Hall, J. Owlett, and S.J.P. Todd. Relations and entities. In G.M. Nijssen, editor, *Modelling in Data Base management Systems.* North-Holland, 1976.

[JM80] S. Jones and P.J. Mason. Handling the time dimension in a data base. In *Proc. International Conference on Data Bases*, pages 65–83, Heyden, July 1980. British Computer Society.

[Kam68] H. Kamp. *On the Tense Logic and the Theory of Order.* PhD thesis, UCLA, 1968.

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM*, 29(3):699–717, July 1982.

[Kro87] F. Kroger. *Temporal Logic of Programs.* Springer-Verlag, 1987. EATCS Monographs on Theoretical Computer Science.

[KSW90] F. Kabanza, J.-M. Stevenne, and P. Wolper. Handling infinite temporal data. In *Proceedings of PODS Symposium*, pages 392–403, 1990.

[Lor87] R.G. Lorentzos, N.A.; Johnson. TRA: A model for a temporal relational algebra. In *Proceedings of the Conference on Temporal Aspects in Information Systems*, pages 99–112, France, May 1987. AFCET.

[Mai83] D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.

[McK86] E. McKenzie. Bibliography: Temporal databases. *ACM SIGMOD Record*, 15(4):40–52, December 1986.

[MS90] E. McKenzie and R. Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 15(2):207–232, 1990.

[MS91a] E. McKenzie and R. Snodgrass. An evaluation of relational algebras incorporating the time dimension in databases. *compsurv*, 23(4), December 1991.

[MS91b] E. McKenzie and R. Snodgrass. Supporting valid time in an historical relational algebra: Proofs and extensions. Technical Report TR–91–15, Department of Computer Science, University of Arizona, Tucson, AZ, August 1991.

[NA89] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(2):147–175, 1989.

[Qui53] W.v.o Quine. *From a Logical Point of View.* Harvard University Press, Cambridge, 1953.

[RKS88] M. A. Roth, H. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational databases. *TODS*, 13(4):388–417, 1988.

[RU71] N. Rescher and A. Urquhart. *Temporal Logic.* Springer-Verlag, 1971.

[SA85] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *Proceedings of ACM SIGMOD Conference*, pages 236–246, New York, 1985. ACM.

[Sar90] N.L. Sarda. Algebra and query language for a historical data model. *The Computer Journal*, 33(1):11–18, February 1990.

[SGM89] R. Snodgrass, S. Gomez, and E. McKenzie. Aggregates in the temporal query language tquel. Technical Report TR–89–26, Department of Computer Science, University of Arizona, Tucson, AZ, November 1989.

[Sno87] R. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[Sno90] R. Snodgrass. Temporal databases: Status and research directions. *ACM SIGMOD Record*, 19(4):83–89, December 1990.

[Soo91] M.D. Soo. Bibliography on temporal databases. *ACM SIGMOD Record*, 20(1):14–23, March 1991.

[SS87] A. Segev and A. Shoshani. Logical modeling of temporal data. In *Proceedings of ACM SIGMOD Conference*, pages 454–466, San Francisco, May 1987. ACM.

[SS88] R. Stam and R. Snodgrass. A bibliography on temporal databases. *Database Engineering*, 7(4):231–239, December 1988.

[SWKH76] M. Stonebraker, E. Wong, P. Kreps, and G. Held. The design and implementation of ingres. *ACM Transactions on Database Systems*, 1(3):189–222, September 1976.

[Tan86] A.U. Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[TC90] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *International Conference on Very Large Databases*, pages 13–23, 1990.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal Databases.* Benjamin-Cummings, 1993.

[TG92] A. Tansel and L. Garnett. On roth. korth, and silberschatz's extended algebra and calculus for nested relational databases. *TODS*, 17(2):374–383, 1992.

[Tuz89] A. Tuzhilin. *Using Relational Discrete Event Systems and Models for Prediction of Future Behavior of Databases.* PhD thesis, New York University, October 1989.

[Ull88] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.

[vB83] J.F.A.K. van Benthem. *The Logic of Time.* D. Reidel Publishing Company, 1983.