PERSPECTIVES IN ELECTRONIC SHOPPING: ON BEYOND AUTOMATED ORDER ENTRY

by

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

and

Tomas Isakowitz
Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10003

December 1989

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-90-8

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Perspectives in Electronic Shopping: On beyond Automated Order Entry

Tomás Isakowitz* Steven O. Kimbrough†

December 25, 1989

Abstract

Large-scale 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. The aim of this paper is to address these requirements for electronic shopping systems. We show how an abstraction (or **isa**) hierarchy with an imposed distance metric can be used as a represetational basis for modeling the salesperson's rôle (as embodied in the surplus and shortage problems) in an electronic shopping system. 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.

key words: decision analysis, decision support systems, electronic shopping, preference modeling, user interfaces, utility theory, multiattribute utility theory

1 Introduction

Both technical developments and economic forces are continuing to evolve in a direction favoring computer- and communications-based services for purchasing activities, either by consumers or by businesses. On the technical side,

*Department of Information Systems, New York University, Stern School of Business, New York, New York, 10003

†Department Decision Sciences, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, USA

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

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 extent of the offerors of these products, viz. 24-hour, worldwide trading of equities; and globalization of commerce continues to accelerate. But whether widespread electronic shopping comes sooner or later, the time is surely ripe for investigating the theory and principles of design for supporting effective electronic shopping. This is true, in spite of such commercially disappointing products as electronic banking and treasury workstations.

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—Is there need for such systems? Is there sufficient willingness to pay for them? Which sectors of the population should be addressed? Which lines of products should be involved?—and so on. There are system issues, such as establishing the various protocols for network communication; developing a formal language for business communication [4,8]; ensuring privacy and correctness of the transactions; keeping the information updated; and building a suitable user interface. 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. 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].

Electronic shopping, on our view, represents the microprocessing descendant of the shopping by mail concept, in which the shopper, at home or at the office, uses a computer to access an on-line catalog, to browse over and inquire about its offerings, and to initiate purchases. Among the advantages of electronic shopping systems are these:

- their ability to provide continually updated, current information
- their potential for substantially lowering shoppers' transaction costs [3,11, 12], including both the costs associated simply with ordering and invoicing, and the costs associated with searching among offerings in a market
- the opportunities they present in the implemention of new marketing strategies (advertisements can be done via the system) and in measuring their effects

The requirements for a successful system for electronic shopping are extensive. Secure and fast communication mechanisms have to be present; the system will need a friendly user interface; reliability and availability standards must be

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

stringent, and so forth. The impetus for many of these elements, however, goes well beyond requirements induced by electronic shopping. Consequently, present technology is sufficiently advanced, in many of these areas, so as not to be an impediment to the development of electronic shopping systems. Consequently, our present focus is on how to provide for what we call the *salesperson's rôle* in an electronic shopping system.

In general, a customer needs guidance and suggestions in order to find what he or she is looking for. The ability to perform these tasks will have, we believe, a substantial effect on the success of electronic shopping systems. A system that presents screen after screen of product descriptions will place on the customer the burden of finding the right item. Furthermore, this will result in a dull, long and boring process that will discourage use of the system. Even presenting the information in a catalog–like fashion with indices will not be good enough. What is needed is an intelligent system that is capable of obtaining from the customer enough information to guide him or her through the shopping process. The system should also be able to take into account customer preferences, which it should be able to learn from succesive sessions. Basically, it has to play the rôle of a good salesperson: understand what the customer is looking for, remember his or her personal preferences and make appropiate suggestions. In addition, it 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 is based in part on the paper by Lee and Widmeyer [9], which proposes a graph representation of the data. There, a search for the item to propose to the customer is implemented as a graph search procedure. We extend their ideas so that the system is capable of dealing with several aspects of the search. We do so by representing *multiple* perspectives on, or attributes of, the items that are for sale. Thus we not only include the general category of the desired object (e.g., a pair of pants) but also other attributes such as for cost, color, fashion, and so forth. Further, we are able to interpret our representation as an encoding of a multiattribute utility function (see §7). 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 deal with two basic problems: *surplus* and *shortage*.

1. Surplus occurs when the customer issues a vague request. For example he asks for a pair of pants. If this is J.C. Penney's system, chances are that there is more than one pair of pants in the store. 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, the season in which the garment will be used, color preferences, the sex of the person that will use it, and so forth.

2. Shortage occurs when a request is issued for an item which is not available. Supose that a customer asks for item number *jcp 279-1730 d*—a pair of

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

pants—and the item 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.

We propose a system to perfom—or at least to approximate—the *salesperson's rôle.* Using graphs as a data structure to represent catalogs and preferences on items in them, we have implemented operations to support the sorts of requests described above. A prototype program (listed in part in the appendices) was written in Prolog. As an example, we used a J.C. Penney's catalog, taking into account attributes such as *Clothing category, Color, Cost, Sex, Fashion* and *Season*. The system was tested by several people and the results were rewarding.

The remainder of the paper is organized as follows. We begin, in §2, with a discussion of the Lee and Widmeyer work and of the basics of the data structure we propose for supporting the saleman's rôle in an electronic shopping system. The Lee and Widmeyer work encompasses only the single perspective (single attribute, unidimensional) case. In §3, we extend the Lee and Widmeyer work for the single perspective case, by elaborating upon their basic data structure. §4 extends our treatment of the problem to the multiple perspective case. We show, in §5, that the shortage, as well as the surplus, problem can be handled in essentially the same manner under our representation scheme, and in §6 we discuss how learning and idiosyncratic preferences can be incorporated into our proposed system. We demonstrate, in §7, 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. Finally, we conclude in §8.

2 The Basic Graph Representation

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 we are faced with is how to organize the information on the items for sale in a way that supports the search process. In general, printed catalogs organize the information into categories. When looking for an item, one determines the category to which it belongs and then flips through the corresponding pages. For example, a catalog for a clothing shop could split 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 one arrives at a hierarchical organization of the different categories that together represent the products for sale.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Different ways of structuring the information are possible (and actually desirable), but the point we want to make here is that such a structure does exist and is inherent to the problem. Lee and Widmeyer [9], using ideas originated in semantic networks, propose representing the structure as a directed graph. The nodes are categories, leaves are individual items, and the directed arcs represent the inclusion relation between categories and are called *isa-links*. A possible graph representing the information about clothes is presented in figure 1.

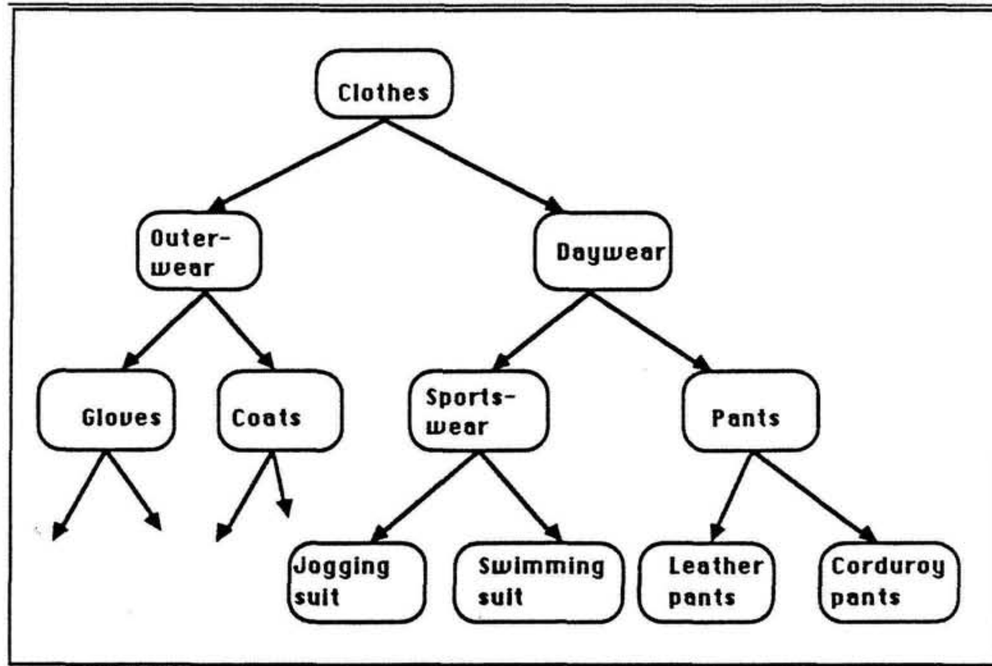

Figure 1: A Graph for Clothes

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. They are groupings used to support the search process. The graph imposes a distance notion, for which a natural measure is 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* available (in stock) item. For example, in response to a request for a sportswear item, a graph traversal is performed which checks whether there are any jogging suits or swimming suits before proposing leather pants.

In [9] a Prolog implementation is proposed and discussed. The isa links

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

are represented by predicates: `isa(Cat1,Cat2)` represents an arc from `Cat2` to `Cat1`. For example, the previous graph is represented by the folowing predicates:

```
isa(gloves, outerwear).
isa(outerwear, clothes).
isa(coats,outerwear).
isa(jogging suit, sportswear).
isa(sportswear, daywear).
isa(swimming suit, sportswear).
isa(daywear,clothes).

    ⋮
```

We also need a way of indicating the codes of the products belonging to the different categories. We do this via the predicate `instance_of(Individual, Category)`. 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. One could actually include the codes in the graph itself, but this is space inefficient. (Since there might be more than one way of structuring the information into graphs, a node might appear in more than one graph. Including the product codes in the graphs would force us to include them in every graph in which their parent nodes appear, thus wasting space by duplicating information. The *instance_of* predicate implicitely includes the product codes as leaves of the graphs, without wasting space.)

We will also use the `instance_of` predicate to specify availability of an item. Thus given that `cat` is a leaf node in some graph, if the query `?- instance_of(Code, cat)` is unsatisfiable, we conclude that there is no availability for items in the category `cat`. So the codes in this paper will represent actual objects, although in a real system other predicates could be present to keep the inventory.

The predicate `match` is used by [9] to find suitable candidates for a user's request:

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

This performs a search of the descendants of a node in order to find an individual of the desired category. For example, in the clothing graph presented in figure 1 in response to the query `?- match(sportswear, X)`, a graph traversal would be activated which first checks for individuals of category *jogging suit*, if none are available *swimming suits* are tried and if there are none of them, the search

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

fails. A natural extension is to continue the search in the subtree rooted at *pants*. Thus the system might propose leather pants, which are not sportswear items, but this may be the best match the store can offer (depending on the graph used). 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, the proposed graph representation and Prolog predicates in [9] are insufficiently powerful to handle certain aspects of the problem.

A) 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 specifying preferences between siblings is through the sequencing in the Data Base, a rather undesirable feature. Notice that this is not due to the Prolog implementation, but to the absence of ways of specifying preferences bewteen siblings in the data structure. Another problem related to this fact occurs when a node has more than one parent. Consider the graph of figure 2. Although exagerated, this example shows the need for specifying a more precise notion of distance between nodes. If a request for a *Swiss Army Knife* were issued and none were available, the system might propose either a *Hammer* or a *Missile*. The graph designer has no control over which should be presented first. We propose a solution to this problem by adding weights to the arcs.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

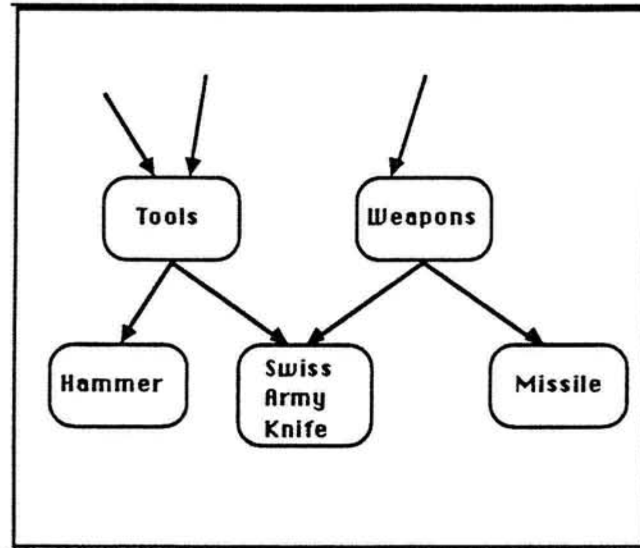

Figure 2: The Swiss Army Knife Problem

B) The proposal in [9] handles shortage by calculations on the graph data structure, but in order to handle surplus, a different approach is taken there. Information about user preferences is introduced in order to focus the search, but in an incompletely specified manner. We propose to represent user preferences in graphs, thereby providing a unified treatment of both the surplus and the shortage problems.

C) In general, there is more than one aspect to take into account when proposing a product to a customer. For example, when buying clothes it is not only important to consider the categories as described in figure 1, but the system should also comply with such other constraints such as cost, color, and fabric. We call these aspects *perspectives* and present a system that is capable of dealing with a number of perspectives simultaneously. Our intention is that the different aspects to be taken into account should be treated as multiple attributes in an underlying multiattribute utility (or value) model [6]. We shall expand upon this point in the sequel.

3 Expanding the Graphs

In this section, we introduce the idea of enhancing the graphs by assigning costs to the arcs. This turns out to solve some of the problems presented in the previous section. We also discuss the implementation of the data structure and the new algorithms.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

3.1 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. 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. Thus, low cost is equated with close similarity, and the cost of an arc has nothing to do with the cost of any item, e.g., leather pants, in the system.

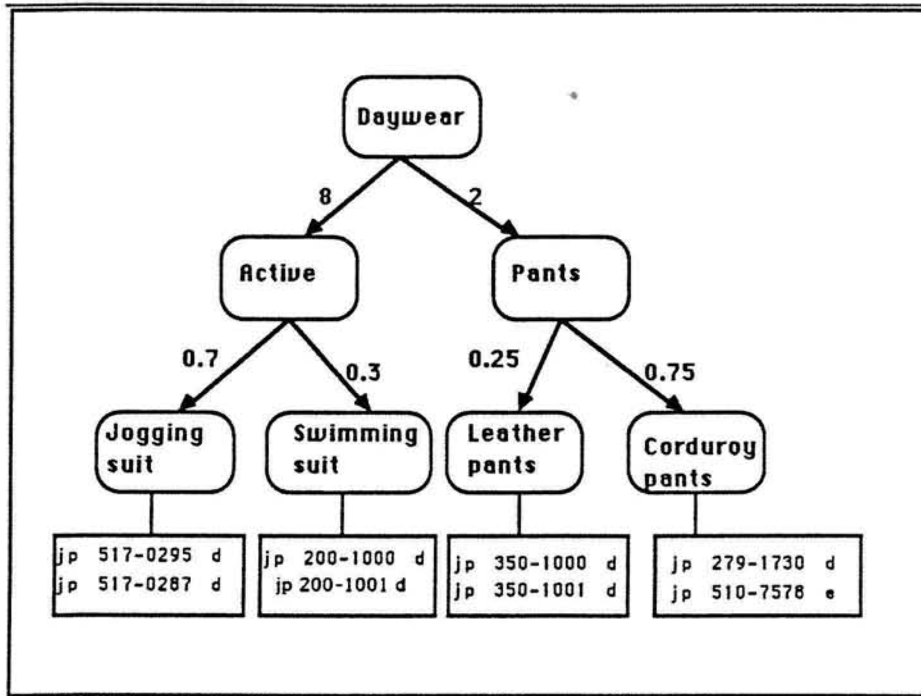

Figure 3: Adding Costs to the Arcs

In figure 3 we present a possible cost assignment for some arcs in the *Clothing* graph. The codes (catalog numbers) appearing inside the rectangular boxes correspond to individuals which are instances of the nodes immediately above them, to which they are linked. Throughout this and the following sections we will use that subgraph as our example, rather than the whole *clothing* graph (figure 1), in order to conserve the space needed for figures and examples.

The question now is how these costs, or arc weights representing similarity, are to be assigned. One way to do so (and in §7 we will consider other ways) is to take into account the probability of chosing a link to an immediate descendant. Since the higher the probability the lower the cost has to be, we use the formula

$$C_{i,j} = 1 - P_{i,j} \tag{1}$$

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

where $P_{i,j}$ is the probability of choosing the arc from node i to node j, given you are at ancestor (higher-level) node i. For present purposes, we assume that this probability is independent of how you got to node i. (This assumption could be relaxed, but computational costs would increase significantly.) Also, we note that assigning probabilities, hence costs, in this way may well be a prudent heuristic for the operator of the system, although other heuristics are certainly sensible, too.

Suppose, for example, that the probability of a customer chosing leather pants when looking for pants is 75 %. The cost associated with that arc will therefore be 0.25. Similarly a probabilty of 25 % for Corduroy pants yields a cost of 0.75. In this way the distance between pants and leather pants is shorter than the distance between pants and corduroy pants. A point to take into consideration with this approach is that the further down a node is in the graph, i.e., the further away from the root, the finer the distinction between its succesors. The distinction between *leather pants* and *corduroy pants* is finer than the one between *outerwear* and *daywear*. Thus the costs associated with the arcs ending in the latter nodes should weigh more than the ones associated with the former. The formula used to compute the cost associated with an arc is to multiply the quantity obtained in equation 1 by k^h where h is the height of the source node. (In the examples that follow, we set $k = 10$.) The cost equation now becomes

$$Cost = (1 - Probability) \cdot k^h \tag{2}$$

The system will be very sensitive to variations in the cost figures, hence they should be dealt with carefully. It is advisable to have a program which interacts with the catalog designer and assigns the costs according to his or her specifications. It is possible to do this with a qualitative technique that does not present the subject with any numbers at all.

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

Table 1: Distances from *Pants*

Let us analyze via an example how the costs influence the distance measurements. Table 1 shows the distances from *Pants* to all leaf nodes. We have only included there the leaf nodes because the system has to suggest one of them. It would not make any sense to suggest an abstract category such as *Clothes* or *Daywear*. In the actual program we will only need the distances from internal nodes to the product codes. This distance is defined to be the distance between

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

that internal node and the leaf of which the product code is an instance of. The nodes in the table are ordered from closest to farthest, thus the first suggestion is *Leather pants*, then *Corduroy pants*, and so forth. Notice that we have succeded in specifying a preference between siblings, actually we have a total ordering which is completely specified by the graph. The *Swiss-Army Knive* problem presented in figure 2 is solved similarly.

The possibility of enforcing stricter distance notions opens the doorway to a number of applications. 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 6. Similarly, if the managment wants to stress one product over other ones, it could do this by lowering the costs associated with the corresponding nodes. As we can see from these two examples, adding costs to the arcs provides flexibility without having to pay the penalty of having to redesign the graphs, or of abandoning the graph representation altogether.

3.2 Implementation

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. Since these links are graph independent and there is no need to associate costs with them, this predicate remains unchanged. The cost of the *virtual* arc linking a node with a product code is taken to be 0. Part of the graph depicted in figure 3 is represented by the following predicates:

```
isa(clothing, leather pants, pants, 0.25).
isa(clothing, corduroy pants, pants, 0.75).
isa(clothing, pants, daywear, 2).
isa(clothing, jogging suit, sportswear, 0.7).
isa(clothing, swimming suit, sportswear, 0.3).
isa(clothing, sportswear, daywear, 8).
```

3.2.1 A note on complexity issues

Given a node, how long does it take to find the closest leaf? The answer to this question is important beacuse it will influence the implementation of the system. We do not want our customer to wait for an hour and a half before receiving an answer, or even 5 minutes. The system should run in real time, responding within seconds, simulating a conversation.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Notice that in general there might be more than one path connecting two nodes. This makes the computation of the distance quite time consuming. There are at least two possible implementation strategies.

1. The system could store internally the graphs and find the closest leaf by transversing the graph with each request. Suposing there are n nodes and e edges in the graph, this approach takes $O(e)$ space and $O(n^2)$ time, using Dijstra's algorithm [1].

2. Another approach precomputes the distances between the internal nodes and the leaves and stores them using $O(n^2)$ space. Although this process is costly $(O(n^3))$, it is a one time cost. Given a node, finding the closest leaf can then be done in linear time. As we explain later on, in some cases the distances from one node to all leaves will have to be retrieved and sorted. This can be done in $O(n \cdot ln(n))$ time.

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 trees, as in the examples so far. In this case there is at most one path connecting any two nodes. Using Dijstra'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 apporach is advisable. 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 to propose a prototype program. In a real system, however, the factors discussed in the last parragraph should be taken into account to decide which strategy to adopt.

3.2.2 The Prolog implementation

We adopt the strategy of precomputing the distances beween the nodes and the leaves and storing them. (Our implementation is in Prolog, so this information is 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. A program to precompute these distances and store them as Prolog facts appears in appendix A. The predicate `dist(Graph, Node, Leaf_node, Distance)` denotes that `Distance` is the distance between `Node` and `Leaf_node` in graph `Graph`.

If there is no path connecting two nodes, the distance separating them is taken to be infinite. We represent the value of infinite with the atom `top` which is greater than any integer. This concept is enforced by adding the clause `dist(_, _, _, top)` after all the other `dist` facts.

In the example we are using, some of these facts are:

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
dist(clothing,daywear,'jp 200-1000 d',8.3).
dist(clothing,daywear,'jp 350-1000 d',3.25).
dist(clothing,'corduroy pant','jp 200-1000 d',12.05).
dist(clothing,'corduroy pant','jp 350-1000 d',1).
dist(clothing,'jogging suit','jp 200-1000 d',1).
dist(clothing,'jogging suit','jp 350-1000 d',11.95).

dist(_, _, _, top).
```

We use the meta-logical **bagof** feature of Prolog which works as follows:

```
bagof(Expression, Predicate, List)
```

returns a `List` whose individual members have the form `Expression` which is built up with the values satisfying **Predicate**. **List** contains no repetitions. For example **bagof(X, man(X), L)** will return a repetition-free list of all `X` satisfying `man(X)`. We also asume the existence of a **sort** predicate that returns as its second argument a lexicographically ordered version of its first one, both of which are lists.

Given a graph `G` and a node `N`, the predicate **reachable((G, N), L)** holds whenever `L` is a list of the leaf nodes of `G` and their associated distances to `N`. The list is in ascending order of distance.

```
dist((G,N),X,Cost):-        dist(G,N,X,Cost).
dist(G,N,X,0):-             instance_of(X,N).

reachable((G,N), L) :-         bagof( (Cost, Node),
                                  ( dist((G,N),Node,Cost),
                                    less_than(Cost, top)),
                                  L1),
                            sort(L1,L).

less_than(X,Y):-            number(X), number(Y), X < Y.
less_than(X,top):-          number(X).
```

In order to interact with the user, the system captures the desired selection from him. This selection consists of a graph and a node, for example **(clothing, pants)**. The predicate **reachable** is set as a goal which returns the ordered list of leaves, i.e., the ordered list of items actually in the catalog. Then a dialog is started which suggests products to the user in order of proximity. The predicate **match** implements this idea:

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

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.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
dialog([]).
dialog([(Cost,Node) | Tail]) :- show(Node),
                                ((more,dialog(Tail));true).

more :- print('Would you like to see more? '),!, read(y).
```

Show displays a description of the product beeing offered. It identifies all the categories to which it belongs (it might be a member of more than one as we'll see later), and displays them using nice_print.

```
show(N) :- bagof(Category,instance_of(N, Category),L),
        nice_print(L),
        write(' with code # '),
        write(N),
        nl.

nice_print([]).
nice_print([X]):-  print(X).
nice_print([X,Y]):-  print(X),print(' '),print(Y).
nice_print([H|T]) :-  print(H),print(', '),nice_print(T).
```

The following is a sample session. A match for *pants* is presented, the system suggests leather pants, corduroy pants, swimm suits and jogging suits in that order.

```
?- 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 ? y.

swimsuit with code # jp 200-1001 d
Would you like to see more ? y.

jogging suit with code # jp 517-0287 d
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
Would you like to see more ? y.

jogging suit with code # jp 517-0295 d
Would you like to see more ? y.
no.
```

4 Perspectives

In general, when looking for a product to buy, we take into consideration more than one aspect of it. For example when looking for a pair of pants we may have in mind certain color and cost preferences. We call these various aspects *perspectives*. In this section we show how certain operations on graphs allow us to deal with more than one perspective.

4.1 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 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 Eucidean 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:

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3}$$

Here $(x_1 - x_2)$ is the distance along the x-dimension and $(y_1 - y_2)$ is the distance along the y-dimension.

To take an example from the world of clothes, consider the *color* perspective represented in figure 4.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

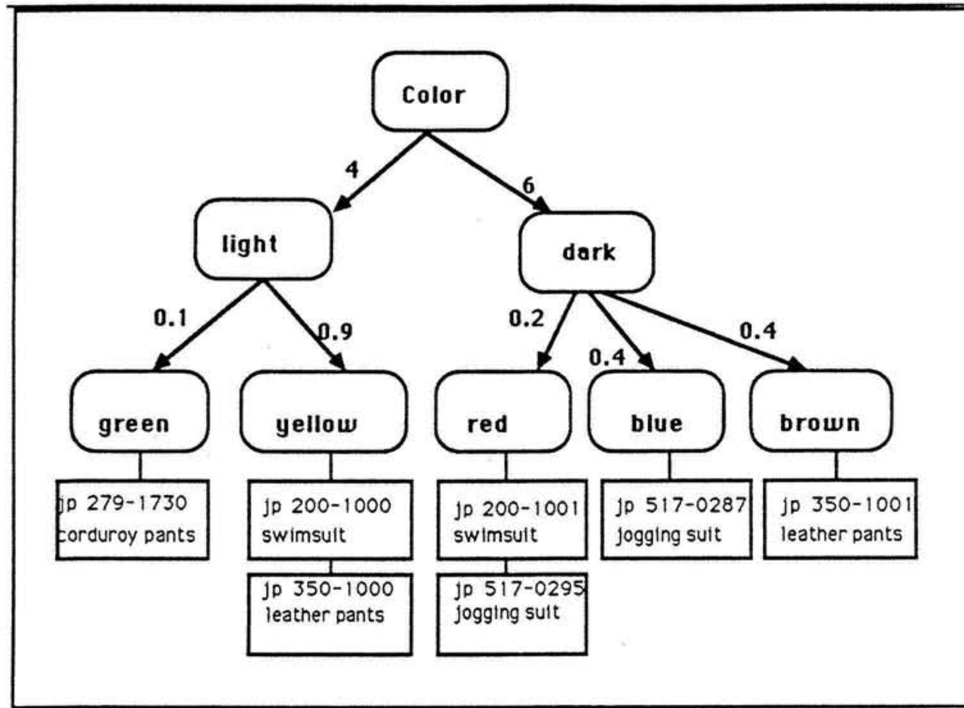

Figure 4: A Perspective for Color

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.

This *color* graph defines a notion of distance different to the one defined by the *clothing* graph. For example, since *jp 279-1730* and *jp 510-7578* are corduroy pants, they are very close in the *clothing* perspective since the distance between them is 0 . From the point of view of *color,* however, their distance is 10.3, thus they are distant from each other. We would like to combine the information present in both graphs to support a complex search. Supose the customer is looking for *blue pants.* Notice that there are not any 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*?

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

4.2 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). As before, the `instance_of` predicate will represent the links between the nodes of the graphs and the codes of the products. 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 in a *PROLOG* implementation of this data base would look similar to the following.

```
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
```

4.3 The multi–perspective distance

Carrying 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*:

$$d((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \sqrt{\sum_{i=1}^{i=n} d(x_i, y_i)^2} \tag{4}$$

alternatively, one can use a *maximal metric*:

$$d((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = max_{1 \leq i \leq n}(d(x_i, y_i)) \tag{5}$$

From a computational point of view it would make sense to find a suitable multi–dimensional distance function that does not require searching through *multidimensional graphs* for shortest paths between nodes. The metrics presented above satisfy this compositionality requirement.

We implement the Euclidean metric with the *PROLOG* predicate `euclid` whose first argument is a list of distances, along different dimensions, between two points and whose second argument is the Euclidean distance computed using equation 4 above.

```
euclid(L,D):-           sum_squares(L,S),
                        D is sqrt(S).
sum_squares([],0).
sum_squares([H|T],D):-  sum_squares(T,D1),
                        times(H,H,Hsquare),
                        plus(Hsquare,D1,D).
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Similarly the maximal–metric is given by:

```
maximal([H],H).
maximal([H,T],D):-          max(H,T,D).
maximal([H|T],D):-          maximal(T,D1), max(H,D1,D).
```

Since we have represented infinite distances by the `top` atom, we have to make sure that the arithmetic operations are performed correctly, i.e. $\texttt{top} + x = \texttt{top}$, $\texttt{top} \cdot x = \texttt{top}$, $max(\texttt{top}, x) = \texttt{top}$, etc. This is done using by `plus`, `times` and `max` predicates:

```
plus(top,_,top).
plus(_,top,top).
plus(X,Y,S):-        S is X+Y.

times(top,_,top).
times(_,top,top).
times(X,Y,P):-        P is X*Y.

max(top,_,top):- true,!.
     % The  cuts here are to avoid backtracking
     % when looking for candidates.
max(_,top,top):- true,!.

max(A,B,A):-          B < A.
max(A,B,B).
```

The formulæ presented so far provide no means for distinguishing perspectives with regard to relative importance. There certainly should, however, exist means for specifying that a certain perspective is more important than another one in a specific search. 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 swimming suit. However, if the color is important because Susan loves blue, then the blue swimming suit constitutes a better match. In the implementation we associate weights with the pespectives when performing a search. The distance along each perspective is multiplied by the corresponding weight prior to calculating the multidimensional distance. Thus, for the first request one assigns more weight, w_i, to *clothing* than to *color*, while the opposite will occurr in the other case. To do this formally, we use a weighted Euclidean formula.

$$d((x_1, \ldots, x_n), (y_1, \ldots, y_n)) = \sqrt{\sum_{i=1}^{i=n} w_i \cdot d(x_i, y_i)^2} \tag{6}$$

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

4.4 The join operation

We implement the multiperspective match with the *join* operation which 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 and a given node. This list is its first argument, the collection of distances is returned in the third one. The second argument is the target node.

```
coll_dist([],_,[]).
coll_dist([H|T],M,[D|L]):- dist(H,M,D), coll_dist(T,M,L).
```

The computation of the weighted distance is done with the **dist** predicated as well:

```
dist([W,GN],M,D):-  dist(GN,M,D1), % the
                    D is W * D1.   % weighted distance.
```

Notice that no change has to be made to the **match** predicate, all the work is performed by the **dist** predicate.

We now show the result obtained using the *clothing* and *color* perspectives in a summarized form.

1. Searching for a blue pair of pants, emphasizing blue:

```
?- 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
red jogging suit with code # jp 517-0295 d
green corduroy pants with code # jp 279-1730 d
yellow leather pants with code # jp 350-1000 d
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
yellow swimsuit with code # jp 200-1000 d
```

Notice that although leather pants are prefered over corduroy pants in the clothing perspective, a pair of the latter is suggested first since blue is closer to red than to brown.

2. The same query, but now the emphasis is on pants:

```
?- 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
blue jogging suit with code # jp 517-0287 d
red jogging suit with code # jp 517-0295 d
yellow swimsuit with code # jp 200-1000 d
```

Here *leather pants* came before *corduroy* ones, but only the red ones. Even with these adjusted weights, a pair of *yellow leather pants* is further away from blue pants than from green corduroy pants.

4.5 The union operation

Suppose someone is interested in buying pants, would like them to be blue, but would accept green. Using the color perspective given earlier, since there are no blue pants, the system would come up with red or brown ones before suggesting a green pair of pants. How can this user's preference be handled?

What if both green and blue were selected and the best of both outcomes is proposed? This type of 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**. Since minimums are taken here, the role of the weights is reversed: a higher weight means less importance.

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

The minimal metric is computed as follows:

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
minimal([D],D).
minimal([H,T],D):- min(H,T,D).
minimal([H|T],D):- minimal(T,D1),
 min(H,D1,D).

min(top,X,X):- true,!.
min(X,top,X):- true,!.

min(X,Y,Y) :- max(X,Y,X).
min(X,Y,X) :- max(X,Y,Y).
```

The example 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 becomes very cumbersome, we remind the reader that, in any commercial-grade application, a front-end user interface should handle dialogs and provide the system with the formal queries.

By analyzing the output we realize that even though 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* which is organized according to which colors are complementary. In this perspective red and green are close toghether as are lilac and yellow. If someone is looking for something red, the system could use both the *color* and the *complementary colors* perspective to make suggestions. Thus if nothing red were available, it would suggest something that would match with red either according to brightness or according to complement. This is naturally done via the `union` operator:

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


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Using this query, the system will find a match for a jacket that also matches red. Thus it could come up with either a brown (using the *brightness* perspective) or a green (*complementary* perspective) jacket if no red one was available.

5 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 occurrs when a specific product is selected by the user (with a specific product code number) that is not available. In terms of our graph representation, we start at a leaf and need to find a closest distinct leaf. It is significant that shortage can be dealt with in much the same way surplus was.

Supose we are looking for a certain item, say *T-shirt jp 522-1635 d*, and there are none in stock. What should the program offer instead? What happens if we issue the query `match('jp 522-1635')`? There is a basic problem: we are not telling the system under which perspective to search. The solution is to have the system find out to which perspectives *jp 522-1635* belongs and perform a join operation. The weights 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.

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. The assignment of weights to the perspectives will reflect how important the color, the cost, etc. are.

If the system were implemented so that the distances were computed from the graphs as needed, the procedure described above would work fine. The same holds if it stored the distances between coded products, for example `dist(clothing,'jp 274-1730 d','jp 350-1001 d',2.3)`.

Since the implementation adopted does not keep the distances between individuals, we offer a different approach. Find all the abstract categories and corresponding graphs to which the individual belongs, assign weights to the graphs and then find a best match using `join`. This is basically the same procedure as described earlier with the distinction that the set of abstract nodes to which the individual belongs is collected. In the example of the T-shirt *jp 522-1635 d* this collection could be {*(clothing, T-shirt), (color, blue), (cost, cheap),* ...}. The implementation could be as follows:

```
match(Ind):-    individual(Ind),
                collect_categories(Ind, CatList),
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
                    get_weights(CatList,WCL),
                    match(join(WCL)).

individual(Ind):- instance_of(Ind,_).


collect_categories(Ind, Catlist) :-
                    bagof( (Graph, Node),
                           (instance_of(Ind, Node),
                            in_graph(Node, Graph)),
                            Catlist).

get_weights([], []).
get_weights([(Graph, Node) |T1],
                [(Weight, Graph, Node) | T2]):-
                    ask_user(Weight, Graph, Node),
                     get_weights(T1, T2).

 ask_user(Weight, Graph, Node):-
                print('What is the weight associated with '),
                print(Graph), print(Node),
                read(Weight).
```

6 Personal Preferences

So far we have described operations on graphs that combine them in such a way that no new graph traversals are needed for the computations. Both the join and the union operations required simple arithmentic operations to be performed, such as sum, product, minimum and maximum, on the distances of the graphs provided as arguments. We now introduce a natural operation that requires graph traversals, since it works by modifying the underlying graphs.

Supose 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 selected weights in the *color* perspective.

We propose the following solution. Build a graph that only holds the arcs whose weights have to be changed. When computing distances on this new graph default to the other one when no information is vailable. That is, if there is an arc between two nodes in the new graph, then use it; if there is not, try finding one in the other graph.

For the example of Susan, we would build a *Susan_colors* graph containing only the nodes *color, light* and *dark* as shown in figure 5.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

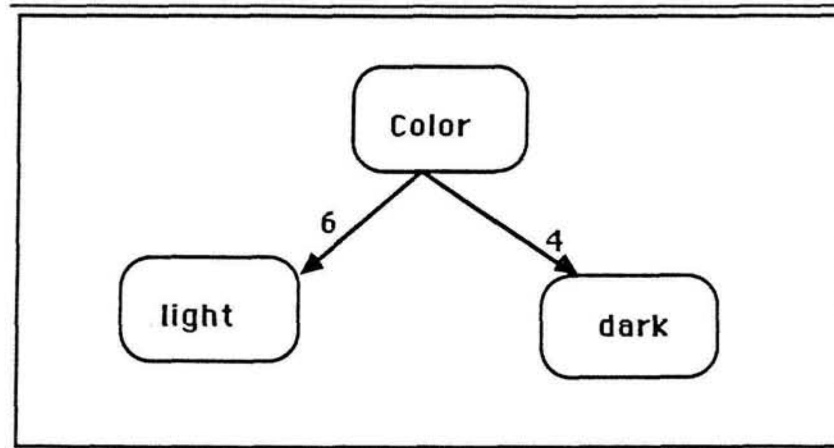

Figure 5: Susan's Preferences

Since it modifies the graph, the default reasoning is implemented via the `isa` predicate:

```
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 infomation is duplicated. Second, it provides for consistency by keeping only one version of the shared information.

The most interesting utilization is probably the representation of personal preferences. By interacting with the user and monitoring his or her choices, the system could observe in which way a user's preferences differ from the stored perspectives. It could then build user models in the form of personal perspectives.

7 Relation 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. Recall that we assigned the cost of the arc $i - j$, $C_{i,j}$, as

$$C_{i,j} = (1 - P_{i,j}) \cdot k^h \tag{7}$$

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

where $P_{i,j}$ is the probability of choosing the arc from (ancestor) node i to (descendant) node j, given you are at node i; h is the height of the arc from the bottom of the graph; and k is a scaling constant that we set to 10 for the sake of the discussion. 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. Also, remember that we have been restricting our discussion to trees, so that there is exactly one path between any two nodes. This assumption could be relaxed, in which case the distance could be measured, e.g., 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 $u(x_j|x_i)$, the utility of going to node j given that you are at node i, as

$$u(x_j|x_i) = \mathtt{max} - d(x_i, x_j) \tag{8}$$

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 8 the distance is a negative linear transformation of the utility, we minimize distance in order to maximize utility.

Given the definition implicit in 8, 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., [5,6,7], 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 path from x_i to x_j,

$$u(x_j|x_i) = \mathtt{max} - (d(x_i, x_k) + d(x_k, x_j)) \tag{9}$$

We call this the *additivity* property. Second, we note an *independence* property: $u(x_j|x_i)$ is independent of the cost of any arc not on the path from x_i to x_j.

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 paths between x_l and x_i and between x_l and x_j, $u(x_i|x_l) > u(x_j|x_l)$. 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.

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:

$$u(x_j|y_1, \ldots, y_n) = \mathtt{MAX} - d(x_j, (y_1, \ldots, y_n)) \tag{10}$$

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

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:

$$d(x_j,(y_1,\ldots,y_n)) = \sqrt{\sum_{i=1}^{i=n}(k_i \cdot w_i \cdot d(x_j,y_i))^2} \quad (11)$$

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 $k_i \cdot \mathtt{max}_i = k_j \cdot \mathtt{max}_j = \mathtt{MAX}$ 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. It is also implementation-specific and can be changed. In particular, we note that with a slightly simpler metric:

$$d(x_j,(y_1,\ldots,y_n)) = \sum_{i=1}^{i=n}(k_i \cdot w_i \cdot d(x_j,y_i)) \quad (12)$$

we have an additive multiattribute utility function, the one most commonly used in practice.

8 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 rôle (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,

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

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.

A Computation of shortest paths

```
help:-
pp("This program calculates the distances from each"),
 nl,
 pp("internal node in a graph to the product codes."),nl,
 pp("To start consult the GRAPH file with the 'isa links'"),
 nl,
 pp("then consult the INSTANCES file"),nl,
 pp("then enter the goal 'tg.'."),nl.

?- help.


% new_member
% new_member(V,L,L1,Lr) is true if L is of the form [L1,V,Lr]
% It is like member, but it also return the left and right
% sublists bounding the element.

new_member(X,[X|T],[],T).
new_member(X,[Y|T],[Y|L],R):-        new_member(X,T,L,R).

% Calculating the distances.
% calc_dist_source(Graph,Source, Nodes,S) computes
% the distances from Source to the nodes in
%        Nodes with paths in graph Graph.
%        It uses Dijstra's algorithm.
%  In S we store the nodes whose distance
%    was already found, Nodes are the ones to consider.
%        It asumes the graph is a tree,
%  hence the first distance computed
%  is final, i.e.
%  there is at most one path berween two nodes.

calc_dist_source(_,_,[],_).
calc_dist_source(G, Source, Nodes, S):-
   member((V1,D1),S),
   arc(G,V1,V,D2),
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
   new_member(V,Nodes,NL,NR),
   D is D1 + D2,
   store(dist(G,Source,V,D)),
   app(Nodes1,NL,NR),
   calc_dist_source(G,Source,Nodes1, [(V,D)|S]).

% store: prints the distances found
%   and stores them if they are leaves.

store(dist(G,Source,V,D)):-
        (leaf(G,V), store_codes(G,Source,V,D)); true.

% store_codes retrieves the product codes corresponding to V.

store_codes(G,Source,V,D):-
 setof(C,instance_of(C,V),Code_list),
 print('Saving '),print(Source),
 print(' to '),print(Code_list),
 nl,print('Distance= '),print(D),nl,
 store_code_list(G,Source,D,Code_list).

store_code_list(_,_,_,[]).
store_code_list(G,Source,D,[H|T]):-
        assertz(dist(G,Source,H,D)),
        store_code_list(G,Source,D,T).


% An arc is a commutative version of an isa.

arc(G,X,Y,C):-        isa(G,X,Y,C).
arc(G,X,Y,C):-        isa(G,Y,X,C).

% Get the set if nodes of a graph.
nodes(G,Nodes):-        setof(N,C^V^arc(G,V,N,C),Nodes).

% A leaf has no descendants.
leaf(G,V):-        not(isa(G,X,V,C)).

%============================================================

comp_dist(G):-        nodes(G,Nodes),
                new_member(X,Nodes,L,R),
                proceed_dist(G,L,X,R).
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
proceed_dist(G,L,X,[]):-        calc_dist_source(G,X,L,[(X,0)]).
proceed_dist(G,L,X,[R1|RT]):-
                app(N,L,[R1|RT]),
                calc_dist_source(G,X,N,[(X,0)]),
                proceed_dist(G,[X|L],R1,RT).

%==========================================================
%        Main level
tg:-
 pp(" Graph name: "),
 read(G),
 pp(" Name of file where to store the computed distances: "),
 read(S),
 comp_dist(G),
 tell(S),
 listing(dist),
 told,
 abolish(dist,4).
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

B The salesman program

```
reachable(H, L) :- setof(        (Cost, Node),
                                 ($dist(H,Node,Cost),
                                  less_than(Cost, top)),
                                 L).


less_than(X,Y) :-
   number(X), number(Y), X < Y,!.
less_than(X,top):-
   number(X),!.

%=-==========================================
% The highest order predicate:

match(Selection) :-
       reachable(Selection, List),
       dialog(List).
%=-==========================================
dialog([]).
dialog([(Cost,Node) | Tail]) :-
 show(Node),
 ((more,dialog(Tail));true).


more :-
 print('Would you like to see more ? '),
 !, read(y).

show(N) :-
        bagof(Category,instance_of(N, Category),L),
        nice_print(L),
        write(' with code # '),
        write(N),
        nl.

nice_print([]).
nice_print([X]):-
      print(X).
nice_print([X,Y]):-
      print(X),print(' '),print(Y).
nice_print([H|T]) :-
      print(H),print(', '),nice_print(T).
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
%==============================================
% Change the distance predicate a little.

$dist((G,N),X,Cost):-        dist(G,N,X,Cost).
$dist(G,N,X,0):-            instance_of(X,N).


% now dist((W,(G,N)),M,D)
$dist([W,GN],M,D):-      $dist(GN,M,D1),
                         D is W * D1.

% now dist(join([[W_1,(G_1,N_1)],
%     [W_2, (G_2,N_2)],...,[W_k,(G_k,N_k)]],M,D)

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

% now dist(union([(W_1,(G_1,N_1)),
%     (W_2,(G_2,N_2)),...,(W_k,(G_k,N_k))],M,D)

$dist(union(EGL),M,D):- coll_dist(EGL,M,L),
                        minimal(L,D).
coll_dist([],_,[]).
coll_dist([H|T],M,[D|L]):-
   $dist(H,M,D), coll_dist(T,M,L).

%===============================================
%                METRICS
%===============================================
euclid(L,D):-          sum_squares(L,S),
                       D is sqrt(S).
sum_squares([],0).
sum_squares([H|T],D):-  sum_squares(T,D1),
                        times(H,H,Hsquare),
                        plus(Hsquare,D1,D).
%===============================================
%        ARITHMETIC

plus(top,_,top).
plus(_,top,top).
plus(X,Y,S)        :-        S is X+Y.
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

```
times(top,_,top).
times(_,top,top).
times(X,Y,P)        :-        P is X*Y.
%============================================
% Maximal metric:

maximal([H,T],D):-         max(H,T,D).
maximal([H|T],D):-         maximal(T,D1),
                           max(H,D1,D).
max(top,_,top):- true,!.
max(_,top,top):- true,!.

max(A,B,A):-        B < A.
max(A,B,B).

%============================================
% Minimal metric:

minimal([H,T],D):-         min(H,T,D).
minimal([H|T],D):-         minimal(T,D1),
                           min(H,D1,D).

min(top,X,X):- true,!.
min(X,top,X):- true,!.

min(X,Y,Y) :- max(X,Y,X).
min(X,Y,X) :- max(X,Y,Y).
```

References

[1] Aho, A., J. Hopcroft, J. Ullman. *The Design and Analysis of Computer Algorithms,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

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

[3] Clemons, Eric K. and Steven O. Kimbrough, "Information Systems, Telecommunications, and Their Effects on Industrial Organization," *Proceedings of the the Seventh International Conference on Information Systems,* Leslie Maggie et al., eds., San Diego, CA, (December 1986), 99-108.

[4] Dewitz, Sandra K., and Ronald M. Lee, "Legal Procedures as Formal Conversations: Contracting on a Performative Network," *Proceedings of the*

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08

Tenth International Conference on Information Systems, Janice I. DeGross, John C. Henderson, and Benn R. Konsynski, eds., Association for Computing Machinery, Baltimore, Maryland, (December 4-6, 1989), 53-65.

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

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

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

[8] Kimbrough, Steven O. and Michael J. Thornburg, "On Semantically Accessible Messaging in an Office Environment," *Proceedings of the Twenty-Second Hawaii International Conference on System Sciences,* IEEE Press, Washington, D.C., 1989, 566-574.

[9] Lee, Ronald M. and George Widmeyer, "Shopping in the Electronic Marketplace," *Journal of Management Information Systems,* **2**, no. 4, (1986), 21-35.

[10] L. Shastri, L., "A Massively Parallel Encoding of Semantic Networks," *Proc. Distributed Artificial Intelligence Workshop,* Sea Ranch, CA, August 1985.

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

[12] Williamson, Oliver E., "The Economics of Organization: The Transaction Cost Approach," *American Journal of Sociology,* **87**, (1981), 548-575.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-90-08