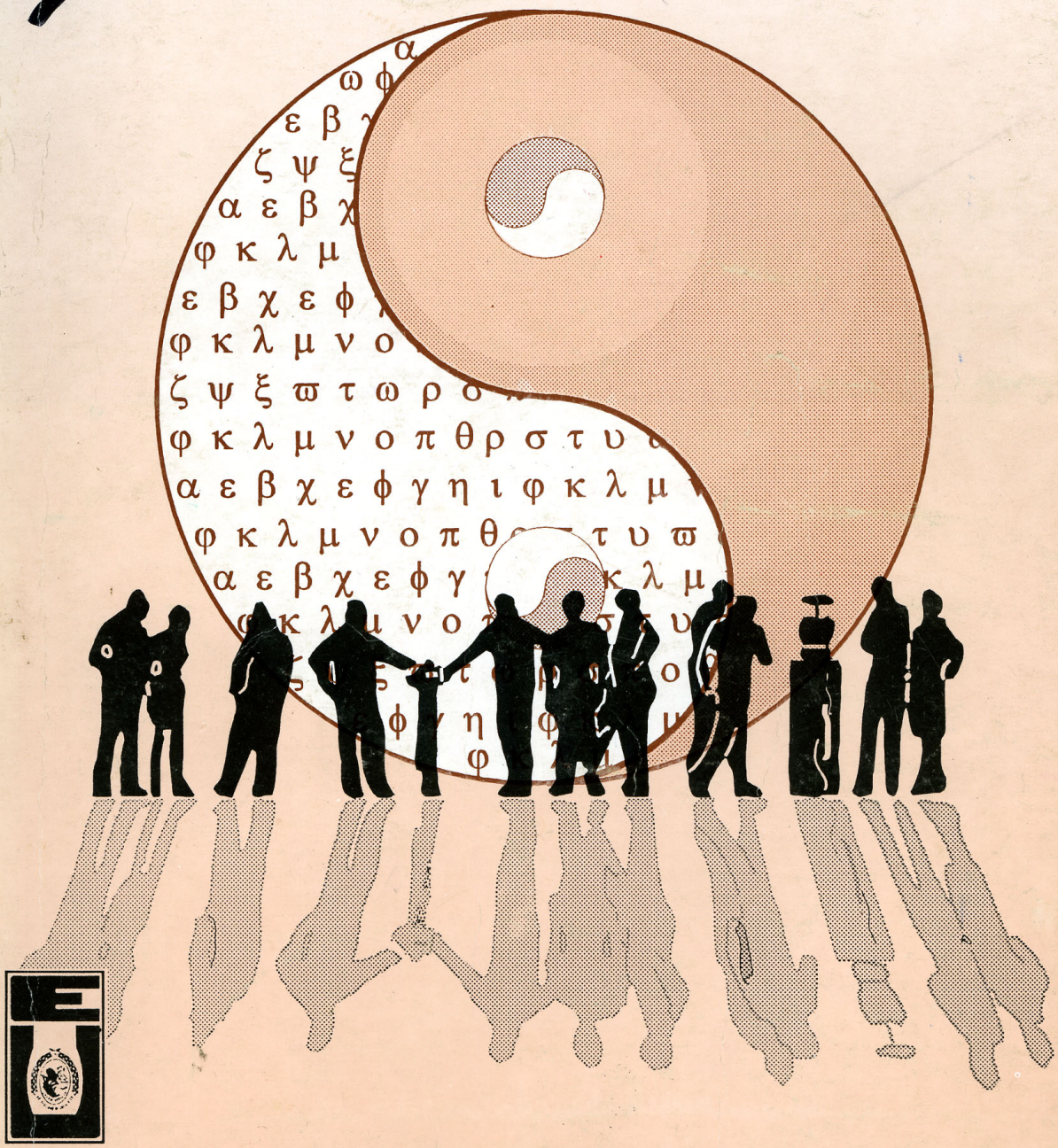
Ingeniería

Revista de la Universidad de Costa Rica
JULIO/DICIEMBRE 1992 VOLUMEN 2 Nº 2



INGENIERIA

Revista Semestral de la Universidad de Costa Rica
Volumen 2 Julio-Diciembre 1992 Número 2

CORRESPONDENCIA Y SUSCRIPCIONES

Editorial de la Universidad de Costa Rica
Apartado Postal 75
2060 Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

CANJES

Universidad de Costa Rica
Sistemas de Bibliotecas, Documentación e información
Unidad de Selección y Adquisiciones-CANJE
Ciudad Universitaria Rodrigo Facio
San José, Costa Rica

Suscripción anual:
Costa Rica:¢750,00
Otros países: US $20.00

Número suelto:
Costa Rica:¢500,00
Otros países: US $10.00

Ingeniería 2 (2): 11-25 1992 San José Costa Rica

MULTIATTRIBUTE DECISION PROBLEMS: AN ALGORITHM FOR LONG MATRICES.

Raúl Alvarado *

RESUMEN

Tratamos aquí con el problema cuando el conjunto de alternativas es muy grande; por lo tanto no es realista usar modelos que buscan en la matriz de decisión más de una vez. Presentamos un nuevo modelo basado en ideas de teoría de utilidad. Finalmente tratamos con el problema donde los datos para la matriz de decisión están esparcidos en varios archivos y lugares. Damos algunas ideas de cómo lidiar con esta situación.

SUMMARY

We deal with the problem when the set of alternatives is very large; hence it is not realistic to use models that browse through the decision matrix more than once; here we present a new model based on the ideas of expected utility. Finally, we look at the problem where the data for the decision matrix is spread across several files and locations. Also here we give some ideas of how to deal with this situation.

1. INTRODUCTION

As human beings we face decision making situations constantly. These decisions may vary in nature and importance, from simple ones, such as deciding whether to watch a movie or to read a book, to more complex and important decisions that may affect us for the rest of our lives, or may change the lives of many persons.

A very important characteristic of the way we make decisions is that we always try to choose the best or optimum alternative. Why choose reading the book instead of watching the movie? Because we think we will have more fun with the book. Why choose this ice cream instead of that one? Because it tastes better. Why buy these shares instead of those? Because we think the former will yield a higher profit.

Besides this optimization tendency, it is also important to notice that "the best" is a subjective value. In the same situation different people make different decisions. One person chooses the vanilla ice cream and another chooses the strawberry ice cream, even though price and quantity are the same.

Any tool for decision making has to consider at least the two characteristics mentioned above: optimization and subjective or personal value. Most of the decision making situations also share a third characteristic: the fact that many problems cannot be solved considering only one aspect of the situation; that is, multiple criteria must be considered.

2. MULTIATTRIBUTE UTILITY THEORY AND THE OPTIMIZATION PROBLEM

Decisions are usually made based on the data contained in a database. Often, the relevant data can be presented to the decision maker as a set of tuples; each tuple represents an alternative. So, the problem for the decision maker is to choose a tuple (or set of tuples) such that it is the best solution to the problem being solved.

* Profesor asociado y exdirector de la escuela de Ciencias de la Computación e Informática.

It is not difficult to show that this process fits very well in the format of Multiattribute Decision Methods (MADM). We said that each tuple represents an alternative, so we can think of the table or relation composed of those tuples as the set of alternatives; we will call this set the feasible set.

Let a be an alternative or tuple and let **F** be the feasible set or table, then the problem of choosing the best solution can be stated as:

Choose "the best" a where $a \varepsilon$ **F**.

We haven't said anything yet about what we mean by the best. We might be tempted to say that a solution is the best if it is preferred over all the other alternatives, but in real life situations this is not a good, workable definition; it has some weaknesses. First, there may be several best alternatives; that is, the solution is not necessarily unique; second, it is not always possible to compare two alternatives in order to decide which one is better; third, we haven't established clearly what we mean by preferred. In order to start solving these problems, it is convenient to rephrase our goal:

Optimize the choice of a where $a \varepsilon$ **F**; this looks like the classical mathematical optimization problem, i.e.: Optimize f(a) where $a \varepsilon$ **F** .

Since a represents a tuple, this means that a is an ordered set of attributes, in the sense of database theory. What is important for us is that a is identified with a set of values, i.e.

$a <-> (a_1,..,a_j,..,a_n)$, hence we can think of α as a point in R^n (here R is the set of real numbers). Hence, the feasible set F is a subset of R^n. Sometimes it is possible to express the feasible set as the set of points in R that satisfy a set of constraints, such as:

Optimize $f(x_1,..,x_n)$ subject to $g_i(x_1,..x_n) <, >=, <=, =, <> 0$. for $i=1,..,m$; where f and g_i are functions from R^n to R.

The function f is called the objective function, and the functions g_i are the constraints. Hence the optimization problem can be stated as:

Optimize $f(x_1,..,x_n)$ where $(x_1,..,x_n) \varepsilon$ F, the feasible set.

If the optimization problem is to maximize the function f, then it can be reformulated as :

Find $\text{Max}\{f(x)\}$, $x = (x_1,..,x_n) \varepsilon$ F.

Often the function f represents a utility function (a multiattribute utility function since $x=(x_1,..,x_n)$); i.e. we want to find :

$$\text{Max}\{u(x)\},\ x \varepsilon F.$$

From a computer science point of view, we think of the tuples or records in the database as the tuples x and the database as the feasible set F. Hence we want that tuple or record that maximizes the utility function for the decision maker.

3. PREFERENCE RELATIONS AND UTILITY FUNCTIONS

The set of real numbers R has a very useful characteristic that is not found naturally in higher powers of R (R^2, R^3, etc). That is the very simple property that, given any two numbers, a and b, you can always compare them; i.e. either $a<b$ or $a=b$ or $a>b$ holds. In other words, R is a totally ordered set (under the canonical order). It is also well known that other sets such as R^2, R^3 or the set of all subsets of a given set don't have this property (under canonical order relations, of course).

When we are modeling a real world situation, and we have to make decisions based on a single parameter such as weight or height then often everything is nice and easy: Just order the objects according to their values for that parameter. Then given any two objects, usually you will be able to tell which one you prefer.

But, when you need two or more parameters or attributes in order to have a good characterization of the objects (e.g., height and weight and color), then, given any two objects, it is not always clear which one you prefer. Bad qualifications in one attribute may be compensated by another (e.g. "the color is ugly but it is the right size"); sometimes this leads to the creation of indifference classes between the objects.

Since we are going to be using the terms preference, order, partial order, etc., it is better to give a precise definitions of all these terms.

Definition: Given a set A, a preference relation on A is a binary relation P on the elements of A. That is, P is a subset of A X A (Cartesian product). If (a,b) is in A then we write aPb (aPb is read as "a is preferred to b").

In general, P is only required to be a binary relation, whose main role (semantically) is to indicate when an element of the alternative set is preferred over another. But, since we are working in R^n, we want our preference relation to be in concordance with the componentwise order relation in R^n. That is, for x and y elements of R^n, $x = (x_1,\ldots,x_n)$ and $y = (y_1,\ldots,y_n)$, we say that $x \leq y$ iff $x_i \leq y_i$ for every $i=1,\ldots,n$. (If $x_i \mid y_i$ for at least some i then we say $x < y$). Hence, we can assume that P is transitive

Definition Let x and y be alternatives. If $x < y$ then we say that y dominates x.
Definition: Let R be a binary relation on A. Let x, y, z be elements of A. The following are properties of R :

Reflexivity For every x in R xR x.
Transitivity If xR y and yR z then xR z.
Antisymmetry If xR y and yR x then x=y.
Symmetry If xR y then yR x.
Asymmetry If xR y then not(yR x).
Connexity For every x and y in R xR y or yR x

If R is reflexive and transitive, then R is called a partial preorder.

If R is reflexive, antisymmetric and transitive, then R is a partial order.
If a partial order (resp. preorder) has the connexity property, it becomes a total or complete (or linear) order (resp. preorder).

Definition: Given a set A (let's call it the set of alternatives) and a preference relation on A, the utility function u on A is defined as a function u: $A \rightarrow R$ such that aPb => u(a) > u(b).

As can be expected, if objects a and b are indifferent then u(a)=u(b).
If a and b are indifferent, then we write aIb. (By indifference we mean some sense of equality; this is also a binary relation and might be used together with the preference relation).

When the elements of the set A are modeled as tuples of two or more attributes, that is $a \leftrightarrow (a_1,..,a_n)$, $n>1$, u is called a multiattribute utility function.

When we know the preference structure on A, we can always define a utility function; obviously it is not unique. Utility functions are just numerical evaluations of preferences; hence they are consistent with the user's preferences (P is a transitive relation). The real problem is that, since A is not necessarily a totally ordered set, then for some alternatives a and b you don't know whether aPb or bPa or aIb.

If we can define this function u, then in some sense we have recreated the nice unidimensional (linear) property of the real numbers. Note that if we have u, then we can extend the preference relation to any pair of alternatives.

The problem in real life situations is that things are not so nice: Either you don't know the preference structure on the set A, or you don't know the utility function u. Some techniques have been developed to deal with these situations [Keeney and Raiffa, 1976]. In order to explain this problem, it is convenient to modify our definition of utility functions:

Definition: Given a set A (let's call it the set of alternatives) and a preference relation on A, the utility function u on A is defined as a function u:A-> R such that aPb <=> u(a) > u(b).

The equivalence symbol indicates that this is a two-way situation. As we said before, if we have a very nice preference relation then the problem is solved. This is also true when you know the utility function. But for some cases you don't know both sides of the equivalence.

Some people have tried to solve the optimizing problem by looking mainly at the utility function. Some of them try to solve some special cases of the utility function, claiming that these cases occur often in practice. It is not rare then to see that they ask for particular properties. "Let's assume that u is concave" or "Let's assume that u is single peaked" or "Let's assume that u is a single variable function" etc., are common prerequisites for this type of method. Once you have the optimum value for u, then it can be traced back to the set A and the preference relation P on A.

Some other methods try to explore more deeply the preference relation itself. That is, they attempt to solve as much as possible of the optimization problem without assuming particular properties of the utility function.

The type of algorithm that we will want to address here is mostly of this second type; even though in some cases we will go to the other side of the equivalence looking for help. We will do this without assuming particular properties for the utility function u. On the other hand, since we have in mind some specific type of problems (optimization for the multiattribute case), this will allow us to be more specific about the utility function.

In order to be more precise about the type of problems and algorithms that we are dealing with, it is convenient to set some notation.

We are going to assume that A is the set of alternatives. A has **m** elements, and each of these alternatives must be evaluated with respect to **n** attributes or criteria. Hence the alternative *a* will be associated with the tuple $(a_1,a_2,\ldots,a_n)$, and we can represent the set A by the matrix $D = (d_{ij})$ mxn. D is called the decision matrix and usually we will denote its components as a_{ij} , since this notation is closer to the notation of the alternatives. In other words, each row of D represents an alternative and each column represents an attribute or criteria. Hence the optimization problem is to choose that row that maximizes the utility for the decision maker.

It is clear that all the attributes do not have the same level of importance. Some or them are more important than others. A very common procedure for dealing with this problem is to assign weights (w_j) to the attributes in order to assess their relative importance. For normalization purposes, these weights are required to be positive and their sum equal to 1.

There are several techniques for assessing the weights and this is a problem worth study by itself. We are not interested in it here; so, we will assume that they are already given.

Before we start dealing with the algorithms, it is necessary to study more closely this matrix D; because depending on the values of m and n, different approaches can be taken. In the next chapter we will show an example of MCDM problems. The last part of the chapter will be dedicated to the analysis of the matrix D when m and n vary.

4. DIMENSIONS OF THE DECISION MATRIX

As we said before, our problem is to choose an alternative among several solutions for a given situation. For example, let us assume that we are in the market looking for software useful for generating Decision Support Systems. After asking several vendors about their products and having listened about the wonders of all of the packages, we end up with a bunch of data. We then arrive at the conclusion that it is necessary to put

some order in all this data before making any decisions. After that, we make the typical tables with all the relevant data.

4.1 Example. DSS Software Products and Vendor

PRODUCT	*AUTHOR/VENDOR*
ABC FPRS	ABC Management Systems, Inc.
Analect	Dialog Inc.
Accent R	National Information Systems
Autotab 300,3000	Capex Corporation
BBL II	Core & Code
Business Modeler	Business Model Systems
Callplan	Calldata Systems
AND 54 PRODUCTS MORE (TOTAL 61)	

TABLE 4.1 DSS Software Products and Vendors.[Reimann and Waren, 1985].

Table 4.1 is just our set of alternatives A. What we would like to have is some sort of order among those 61 alternatives. We really don't need to know, for example, that alternative 20 is 4 times better than alternative 12; we only need to have some ranking. The way this ranking is built depends to some degree on our own personal preferences. Given the same set of data, two decision makers very likely will build different rankings. If you feel confident enough to establish the ranking, then you don't need any of the algorithms to be presented later. If you are not sure about how to build the ranking, then you can choose a suitable procedure among the ones available.

The next step is obviously to look at the characteristics of all these products. Some of the characteristics will be present in all the alternatives with different degrees or levels. After some thought we come up with a set of criteria for evaluating all the alternatives. Sometimes we can group the criteria into several categories according to their similitude. This is shown in the following table:

MODELING
Multidimensionality
Nonprocedurality
Procedural Logic
Simultaneous Equations (detection and solution)
Consolidation and allocations (multidimensional)
Mathematical and financial functions
User-defined functions
Time as a special dimension
Currency conversion and size restrictions

USER FRIENDLINESS
Consistent, natural language commands
Command abbreviations
Help command and clear-error messages and 7 more subcriteria

ANALYSIS
What if
Sensitivity
and 4 more subcriteria

FORECASTING AND STATISTICS
Time as a special dimension
Multiple regression and 5 more subcriteria

DATA MANAGEMENT 6 more subcriteria

COMMUNICATION LINKAGES 4 more subcriteria

COMMAND LANGUAGE 3 more subcriteria

REPORTING 5 more subcriteria

GRAPHICS 7 more subcriteria

VENDOR SUPPORT 13 more subcriteria

COST FACTORS 6 more subcriteria

HARDWARE AND OPERATING SYSTEMS CONSIDERATIONS 4 more subcriteria

TABLE 4.2 User-Oriented Evaluation Criteria for DSS Software.
[Reimann and Waren, 1985].

From Table 4.2 we can see that we are dealing with 77 criteria or attributes grouped in 11 categories. We can think of our decision matrix *D*as composed of 61 rows and 77 columns. As we will see later, the algorithms are especially sensitive to the number of attributes or columns. Hence, 77 attributes are too many for this example. One common way of solving this is to consider the 11 categories as the attributes and assign to each of them the sum of the weights of the former attributes contained in that class. This will reduce our decision matrix to a 61X11 matrix.

Given the matrix D, a wise thing to do is to try to reduce the number of alternatives using the idea of dominance, because elimination of dominated alternatives will help to get rid of useless data and will reduce the size of the matrix. The remaining set of nondominated alternatives sometimes is called the efficient set or the Pareto-Optimal set. Before attempting to apply any algorithms to the nondominated matrix, it is convenient to have some idea of how big it can be; that is: Given the decision matrix D of size mxn, what is the expected number of nondominated alternatives?.

4.2 The number of rows

The following reasoning was developed by Calpine and Golding [1976].

Assume that the entries of matrix D are random numbers distributed uniformly from a given range. Look at the numbers of the n^{th} column, and sort the rows of D such that the numbers in this last column are in decreasing order, i.e. the biggest number will be in row 1. Since we are dealing with random numbers, the probability of any row of being ranked in the r^{th} place is 1/m.

Now let us define p(m,n) as the probability that a row chosen arbitrarily from m rows is nondominated with respect to n attributes.

Given the r^{th} row, the ordering established by the last column ensures that none of the rows below it dominates this r^{th} row. Hence the row is not dominated if and only if it is not dominated by the first r rows (including itself, for simplicity of the notation) with respect to the first (n-1) attributes. According to the definition of p(m,n), then the probability of a row of being nondominated and being the r^{th} is p(r,n-1)/ m.

The probability of an arbitrarily chosen row of being nondominated is

$$p(m,n)=p(\text{U row r is nondominated})=\sum_{r=1}^{m} p(r,n-1)/m$$

$$p(m,n)=1/m[\sum_{r=1}^{m} p(r,n-1)]=1/m[\ p(m,n-1)\ +\ \frac{m-1}{n-1}\sum_{r=1}^{m-1} p(r,n-1)]$$

$$= 1/m\ [p(m,n-1)\ +\ (m-1)\ p(m-1,n)]\ .$$

Let a(m,n) be the expected number of nondominated alternatives, that is a(m,n) = m p(m,n). Substituting in the previous expression we get a(m,n)=a(m, n-1)/ m+a(m-1,n).

This is a recursive definition, using as initial conditions a(m, 1)=a(1, n)= 1 it can be used to calculate a good approximation of a(m, n) as:

$a(m,n)=1+\ln(m)+\ln^2(m)/2!+\ldots+\ln^k(m)/k!\ +\ +$ ß $\ln^{n-2}(m)/(n-2)!\ +\ln^{n-1}(m)/(n-1)!$, where ß is Euler's constant (~ 0.5772).

The following table shows the values of a(m, n) for some m and n values:

m\n	2	4	6	8	10	12
10	3.3e+0	6.9e+0	9.2e+0	9.9e+0	1.0e+1	1.0e+1
100	5.6e+0	2.8e+1	6.1e+1	8.5e+1	9.6e+1	9.9e+1
1000	7.9e+0	7.7e+1	2.7e+2	5.5e+2	7.9e+2	9.2e+2
10000	1.0e+1	1.6e+2	9.1e+2	2.6e+3	5.1e+3	7.3e+3
100000	1.3e+1	3.1e+2	2.4e+3	9.9e+3	2.6e+4	4.7e+4

Table 4.3 Values for a(m, n)

The analysis of this table tells us how critical is the number of attributes in these cases. Note that, in any given row, the value of the entry rapidly grows to the value of m. This is consistent with the fact that, for n large enough, the formula for a(m,n) is very similar to the polynomial approximation of the exponential function evaluated at ln(m), that is a(m,n) ~ exp(ln(m)) = m. This means that for these cases practically all the alternatives are nondominated.

An important conclusion that can be made from this analysis is that, independent of what algorithm is used, a crucial first stage in the procedure is to do a careful selection of the attributes to be employed. This is especially true when m is large, as can be seen in the bottom row of the table.

On the other hand, even if n is small, large values of m are particularly deadly for some algorithms. For example, some of the best algorithms are based on linear programming (e.g., LINMAP). But the simplex algorithm is very sensitive to the number of rows. A reasonable solution for some cases can be

solving the dual linear programming problem, instead.

In Section Five, for the cases when m is very large, we present an algorithm that uses some ideas of expected utility.

5. AN ALGORITHM FOR LONG MATRICES.

Here we are interested in cases when the decision matrix D is very long, that is, when the set of attributes is small and the number of alternatives is really large. Just to have an idea, let us imagine that we are dealing with 20 attributes and one million alternatives. From Table 4.3 we know that the number of undominated alternatives is very big. But even if we know that 200,000 alternatives are dominated, it is unrealistic to think that we are going to browse through this huge table, discarding the dominated ones, and then go back to start applying the standard procedures for small tables or matrices. It is necessary to devise new procedures that browse through the table only once.

5.1 Some definitions

To be able to do that, we must have in advance some knowledge of what we might expect to find in the table. A simple, practical approach to this task is to create some statistical information at the same time we create the decision matrix. Specifically, as we already did, for each attribute j we can define M_j as the maximum value of attribute j for all the tuples represented in the decision matrix D. Similarly we can define m_j as the minimun. That is,

$$M_j = \text{Max}\{a_{ij}\}\, i=1,..,m. \quad m_j = \text{Min}\{a_{ij}\}\, i=1,..,m.$$

Also we need to create, for each attribute j, a frequency record. For this, the range of the domain of j must be divided conveniently. We don't deal with this problem here, since that is a statistical problem out of the scope of our purposes. We just assume that we will be able to know $P(v \leq X_j)$ for a random variable X_j and v a value in the domain of attribute j. If these provisions are made at the creation of the table, then whenever a tuple is inserted, deleted or modified the corresponding values of M_j, m_j and the frequency records are updated.

Usually some of attributes are negatively oriented (e.g. cost, since the smaller the cost the better if all other conditions remains the same). We are going to assume that the attributes are positively oriented, that is, bigger means better. We can do this without loss of generality, and this way the mathematical expressions will look simpler.

It is clear that no tuple can be better than $(M_1, M_2,\ldots,M_n)$, and no tuple can be worse than $(m_1,m_2,\ldots,m_n)$. Note that, even if we don't assume that the attributes are positively oriented, these two hypothetical tuples can still be constructed, but the notation would be more cumbersome.

NOTATION.

- $D = D(a_{rj})$ is the decision matrix. D is a mxn matrix of real numbers or, for any particular column, the data can be taken from a linearly ordered set.
- We will call $(M_1, M_2,\ldots,M_n)$ the ideal tuple, since nothing in the table can be better than it, but maybe there is no tuple that matches its values.
- Let w_j be the weights associated with the attributes.
- X_j are random variables defined over the domains of the respective attributes.
- X is a composite random variable $X = (X_1, \ldots, X_n)$
- $P_j(X_j > v)$ is the probability that the random variable X_j is bigger than v.

We know that if a tuple is better, in each component, than another tuple, then the former is preferred over the latter, but so far we don't know anything else about the preference relation. The ideal tuple can be used to extend the preference relation for other pairs of tuples. Given two tuples, the one "closer" to the ideal tuple should be preferred over the other. For doing this, we need

to define clearly what we mean by closer; that is, we need to define a distance function. A natural distance function for this case is the weighted city distance function.

Given $a=(a_1,..,a_n)$ and $b=(b_1,...,b_n)$, d(a,b) is defined as:

$$d(a,b) = \sum_{j=1}^{n} w_j \mid a_j - b_j \mid$$

To prove that indeed d is a distance function is very simple and will be omitted. Now we can extend the preference relation as follows:

a*P*b iff d(a, ideal tuple) ≤ d(b, ideal tuple). Hence, given a tuple a_r, an arbitrary tuple X is preferred over a_r (XPa_r) iff

$$\sum_{j=1}^{n} w_j \mid M_j - X_j \mid \leq \sum_{j=1}^{n} w_j \mid M_j - a_{rj} \mid \quad <=>$$

$$\sum_{j=1}^{n} w_j M_j - w_j X_j \leq \sum_{j=1}^{n} w_j M_j - w_j a_{rj} \quad <=>$$

$$\sum_{j=1}^{n} -w_j X_j \leq \sum_{j=1}^{n} - w_j a_{rj} \quad <=>$$

$$\sum_{j=1}^{n} w_j X_j \leq \sum_{j=1}^{n} w_j a_{rj} \quad <=>$$

$$\sum_{j=1}^{n} w_j (X_j - a_{rj}) \geq 0$$

Let's assume that we have already browsed through the first part of the file. Let ai be the best tuple that we have found so far; let's call it the incumbent alternative. This tuple ai represents some utility for the decision maker. The next obvious question is if there exists some tuple X, ahead in the file, such that the utility of X surpasses the utility of the incumbent *plus* the cost of searching for X. That is, we want to find a tuple X such that

$$u(X) - C(X) \geq u(ai).$$

It is convenient to discuss here some aspects about the different moments when the values are calculated: at the present moment we know ai and the utility it represents to the decision maker, regardless of the cost already incurred in establishing ai as the incumbent solution. That is, once we reach ai we must forget the corresponding cost *if* we are going to compare its utility against the potential utility of a future tuple X. The utility of X is something that lies in the future and may or may not occur, i.e. we are talking about expected utility. So, what is the present or actual utility of X? Let's define *U* as follows:

$U(X,a_i) = P(XPa_i)\, u(X) - C(X)$.
Note that for a_i $U(a_i,a_i) = u(a_i)$.

This notation for *U* emphasizes the fact that the value of *U* at X depends on the incumbent alternative. This value can be used as a basis for the new algorithm, which can be stated as:

5.2 Algorithm

1. Calculate $U(X,a_i)$ (later we will show how it can be calculated).
2. If $U(X,a_i) \leq u(a_i)$ then STOP; a_i is the "rational" solution.
3. If $U(X,a_i) > u(a_i)$ then fetch the next tuple to be examined. Let's call it the candidate alternative a_c.

   3.1 If ac *P*Pai then replace ai by ac (i.e. ai ...ac) go to step i
   {Since we know that whatever the function u is, it must be consistent with the preference relation}else go to 3.

Step 2 can be modified according to other considerations. If the decision maker is a "risk prone" person, he might decide to continue the search anyway, if the risk is within his limits of tolerance. A similar situation can occur in step 3 if the decision maker has risk aversion.

5.3 Calculation of U(X,ai).

$U(X,a_i)$ has three components: $P(XPa_i)$, u(X) and C(X). First notice that

$$P(XPai) = P(\sum_{j=1}^{n} wj(Xj - a_{ij}) \geq 0).$$

Later we will show how to calculate it.

About the cost function, we also will postpone its analysis since it may depend on several factors, for example, human costs, time costs, use of the facilities, pulling the tuple together if it is stored at different places, etc.

Calculation of u(X) presents two problems. First, we don't know the function u. In some sense this is good, because then we can plug in the formula for *U* any function u that the decision maker wants to use. This also allows different persons (i.e., functions u) to use the system. Each one simply plugs in his/her own personal utility function u. Also, in this sense the procedure is not based on any assumptions about characteristics of the utility function and hence is very general in its scope.

Another possibility is to define here the utility function (which could be my own personal utility function) as :

u(b) = -d(b, ideal tuple) + any constant K

The second problem is that, even if we know the formula for the function u, we still don't know the value of X (and we will never know it, since X is a composite random variable). There are several possible ways to approach this problem, all of them based on the idea that X is a "future" value, something that represents all the tuples to come. Hence, X can be thought as a typical averaging value. Since we have statistical information about each attribute j, we may then substitute the random variable Xj by the mean σ_j of the distribution for attribute j. Other possible candidates for this substitution are the mode or the median for each attribute j. The choice can be made according to each particular case where the algorithm is applied.

5.4 Calculation of $P(XPa_i)$

We said above that whenever the file (or files) is updated we also assume that the frequency table or histogram for each attribute is also updated. For expressing these ideas more clearly let us define some notation: When the database is created n attributes are identified. For each attribute j a range domain Dj is created (these are not the actual mininum and maximal values for the attribute, but the theoretical minimum and maximal allowed values). This range must be partitioned into a suitable number h_j of slots Let k_j be the kth slot and let $f_j(k_j)$ be the frequency for that slot. Hence $P_j(X_j,k_j)$, the probability that the random variable Xj is in the kth slot, is defined as $P_j(X_j,k_j) = f_j(k_j)/m$.

Notation: Let $T_n = \sum_{j=1}^{n} w_j a_{ij}$

We said before that

$$P(XPai) = P(\sum_{j=1}^{n} wj\,(Xj - aij) > 0).$$

When we have only one attribute this formula becomes:

$$P(XPa_{i)} = P(w_1(X_1 - a_{i1}) \geq 0) =$$

$$P((X_1 - a_{i1}) \geq 0) = \sum_{k1=L1}^{h1} P_1(X_1,k_1).$$

Here L_1 is the slot number where $T_1/w_1 = a_{i1}$ was counted.

Notation: Let $L_1 = \text{floor}(T_1/W_1)$ (i.e. the integer part or floor of the number).

For n = 2 we have to find $P(E_2)$. Where E_2 is the event

$$E_2: w_1X_1 + w_2X_2 \geq w_1a_{i1} + w_2a_{i2} (= T_2).$$

A very important note. We are going to assume that the random variables are independent. Conceptually this makes sense, since the attributes should be independent; otherwise we can express some of them through others

and reduce the number of attributes. We have already seen that reducing the number of attributes is very important.

For E_2(see Fig. 5.1), when $X_1 = k_1$ then $X_2 = (T_2 - w_1k_1)/w_2$. Hence k_2 should start at $L_2 = \langle (T_2 - w_1k_1)/w_2$.

$$P(E_2) = \sum_{k1=L1}^{h1} \sum_{k2=L2}^{h2} P_1(X_1, k_1)\, P_2(X_2,k_2)$$

Let us do it once again for n=3.

$E_3: w_1X_1 + w_2X_2 + w_3X_3 \geq w_1a_{i1} + w_2a_{i2} + w_3a_{i3} (= T_3)$.

For E_3, when $X_1 = k_1$ and $X_2 = k_2$ then $X_3 = (T_3 - w_1k_1 - w_2k_2)/w_3$.

Hence k_3 should start at $L_3 = \langle (T_3 - w_1k_1 - w_2k_2)/w_3)$.

$$P(E3) = \sum_{k1=1}^{h1} \sum_{k2=1}^{h2} \sum_{k3=L3}^{h3} P_1(X_1,k_1)\, P_2(X_2,k_2)\, P_3(X_3,k_3).$$

(See Fig. 5.1 at the end). So, the general formula is:

$$P(E_n) = \sum_{k1=1}^{h1} \sum_{k2=1}^{h2} \sum_{k3=1}^{h3} \ldots\ldots\ldots \sum_{kn=Ln}^{hn} \prod_{j=1}^{n} Pj(Xj,kj), \text{ where}$$

$$L_n = \text{floor}((T_n - w_1k_1 - w_2k_2 \ldots\ldots - w_{r-1}k_{r-1})/w_n)$$

.

We can prove now this formula inductively. For (n+1) we have:

$E_{n+1}: w_1X_1 + + \ldots + w_nX_n + w_{n+1}X_{n+1} - w_1a_{i1} + + \ldots + w_na_{in} + w_{n+1}a_{in+1} (= T_{n+1})$.

Now, let's shift the index: sh(j) = j-1.

$E_{n+1}: w_0X_0 + w_1X_1 + \ldots + w_nX_n - w_0a_{i0} + w_1a_{i1} + \ldots + w_na_{in} (= T_{n+1})$. (see Fig. 5.2)

Now $P(E_n+1) = \sum_{k0=1}^{h1} P(E_n)\; P_0(X_0,k_0)$,

where L_0 = floor $(T_{n+1}/w_0$. Since we have already used woko "units" from T_{n+1} then $P(E_n)$ must be calculated for $T_{n+1} - w_ok_o$.

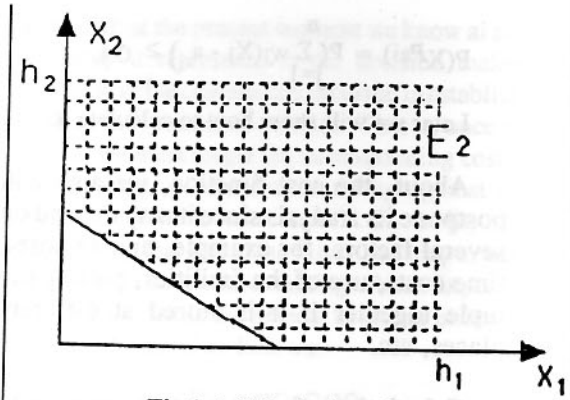


Fig 5.1 Calculation of P[E$_2$]

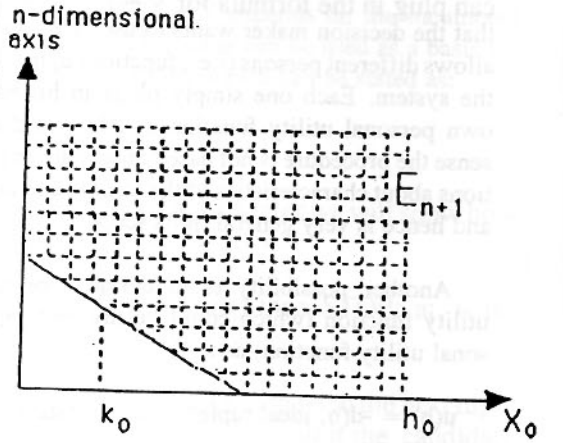


Fig 5.2 Calculation of P[E$_n$]

$$P(E_n+1) = \sum_{k0=1}^{h0} \sum_{k1=1}^{h1} \sum_{k2=1}^{h2} \ldots\ldots \sum_{kn=Ln+1}^{hn} \prod_{j=0}^{n} P_j(X_j,k_j)$$

Now,let us reverse the shift index: $sh^{-1}(j) = j+1$.

$$P(E_n+1) = \sum_{k1=1}^{h1} \sum_{k2=2}^{h2} \sum_{k3=3}^{h3} \ldots\ldots \sum_{kn+1=Ln+1}^{hn+1} \prod_{j=1}^{n+1} P_j(X_j,k_j)$$

5.5 About the cost function

The calculation of the cost of alternative X may be influenced by several factors. Some of them would be technical, such as use of long

distance networks or pagination. Other costs may involve salaries, royalties, etc. All this implies that we cannot devise, in advance, a formula for the cost function valid for all situations. In this sense this will always be an "open" formula; that is, it must be modified for each particular case.

It is important to remark here that we are not deciding whether or not to retrieve a record, based on the cost of the information; this was considered in the first part of the general formula. What we want to do is to calculate the cost of that information. One other obvious fact, but very important to remark, is that the costs must be measured with the same type of units with which the expected utility is measured.

It is convenient to consider several phases in the cost analysis. The first phase should be to devise an strategy for the file search. The second phase should be some procedure to aggregate some "local" cost; and the third phase should be some considerations about how to calculate those local costs. In order to understand this procedure it will be helpful to state, as clearly as possible, the type of scenario that we have in mind. We are considering the case when the decision matrix *D* is physically spread over several files and in different locations; for example, the data for several of the alternatives might be in file1, the data for other alternatives might be in a distant file2, the data for *some* of the attributes for *some* other alternatives might be in file3, the data for the complementing attributes might be in file4, etc.

The decision about where to start the search may be another multiple criteria decision making problem itself. So far, we can only give some common sense heuristics such as:

- Start the search at those files most likely to contain "good" records.
- In case of ties, choose first the file cheaper to get.
- If some records are spread across several files, it doesn't make sense to retrieve the information for one record at a time. Each time you scan some of those files, try to retrieve the information for the maximum number of records possible. This idea will be more clear after reading the description of the third phase.

For the second phase we can build a simple model to help us to keep track of all the different costs.

As we said before, the data for the decision matrix might be spread in several files and also in several forms. Since each of these possible combinations may have its own "local" cost function, it is better to create a table that looks more or less like:

Alternative\File	file1	file2	filef	fileF
a_1	n	0	0	0
a_2	n	0	0	0
a_3	n	0	0	0
a_r	n_{r1}	n_{r2}	n_{rf}	n_{rF}
a_m	n_{m1}	n_{m2}	n_{mf}	n_{mF}

where n_{rf} = number of attributes for alternative r that can be found in file f. Note that, for each row, the sum of all its entries must be equal to n, that is:

$$\sum_{k=1}^{F} n_{rk} = n$$

At first glance this matrix looks very large and indeed it is; but there will be large groups of rows which are identical; hence, the information can be stored using small amounts of memory. For example, assume that we have 15,000 records, with 10 attributes each, distributed over 5 files as follows:

Alternative\File	file1	file2	file3	file4	file5
1	10	0	0	0	0
2	10	0	0	0	0
4000	10	0	0	0	0
4001	0	3	4	0	3
10000	0	3	4	0	3
10001	0	3	0	7	0
15000	0	3	0	7	0

This information could be stored in several forms; for example:

[1...4000] $\leftrightarrow$ (10,0,0,0,0)
[4001...10,000] $\leftrightarrow$ (0,3,4,0,3)
[10001... 15,000] $\leftrightarrow$ (0,3,0,7,3).

Let c_{rfj} be the cost of retrieving the data for attribute j for alternative r from file f. For this aggregation phase we can make two reasonable assumptions: First, for a given file, we can assume that the cost per attribute is the same for all attributes; that is:

$$c_{rfj} = c_{rf}, \text{ for all j in file f.}$$

Note that this assumption is not strictly necessary, but it makes the formulas simpler; c_{rf} is then the potential cost of retrieving the information for record r for any attribute from file f. We call this the potential cost since for some records the actual cost may be zero; for example, if it is only possible to retrieve records in lots of certain minimum size. In these cases, there is some cost for retrieving the first record of the lot (the cost of the lot), the rest are free

The second assumption is that any given file contains the same number ($\partial 0$) of attributes for each record that the file provides information. That is, in a given column, all non-zero entries are equal.

We can then calculate c_r, the cost of retrieving record r as:

$$c_r = \sum_{k=1}^{F} n_{rk} \cdot c_{rk}$$

The third phase of the cost analysis deals with the problem of how to calculate these local c_{rf} cost functions. Here we are focusing our attention on a given file. As in the previous phases, here we may have many factors, such as if the file is online, if it is a local file or if we have to use some network facilities to access it, page sizes handled by I/O devices and operating systems, etc.

I think that the idea of economical lot size is valid in most of the situations mentioned above; we can approximate the cost function with a linear function such as:

$C(r) = K + u \cdot r$ for $1 \leq r \leq LS$ (LS is the lot size).

The value of K is some set up or initial cost and u is a marginal cost. In these cases, the value of c_{rf} would be:

$c_{rf} = C(LS)$ if r is the first record of the lot, 0 otherwise.

5.6 Summary of results:

We showed a formula for calculating the value of $U(X,a_i)$. Based on the values of this formula we constructed the algorithm shown later. Since the formula is composed of several parts, in the next sections we showed how to calculate the different parts that constitute the formula for $U(X,a_i)$.

6 Conclusions

We mentioned the existence of many methods for dealing with multiple criteria decision problems. This fact tells us that this is a far from solved problem. Several factors intervene to create this situation. First, subjective preferences play an important role in these problems, hence the acceptable solutions may vary for different persons. Second, we don't have a "natural" linear or total order for R^n ($n > 1$). Since we can build several orders then we can have different solutions. Third, there is an important difference between the methods for multiple attribute decision making and some other classes of methods such as linear pro-

gramming or sorting files. In these cases we design a method to find the solution, but it is clear what the solution is. By contrast, in MADM we might say that we design the solution; this is so, since is not always clear what the preference relation is.

Mathematicians and other scientists have been studying this type of problems for many years. Some progress has been made; but unfortunately many of the methods developed are theoretically sound but not very useful in practical situations; this is mainly due to long and complicated calculations and checking the truth status of some conditions.

The existence of many methods for MCDM, and the wide variety of them, is also due to the fact that they assume different types of information as input for the models. Some of these methods require long and tedious calculations and questioning sessions with the decision maker in order to gather information about his/her particular preferences. Often these questions are about hypothetical and strange situations. Theoretically, these methods could yield good results; in practice they are seldom used, precisely because they are very impractical.

Lately the use of MADM has been closely related to DSS and Expert Systems; hence it seems that as these two type of systems are becoming more and more important, the same will be true for MADM. The computer implementation of MADM, within the context of DSS or ES, must take care of some problems that did not exist when scientists were dealing only with theoretical models. These are basically problems related to information retrieval. Usually information has an associated cost (computational costs) due to retrieving; in some cases this might affect what the final solution is. This may be specially true for cases where the marginal utility is small if compared with the marginal cost. Hence, these factors should be incorporated in the MADM. The model we presented in section 5. tries to accomplish this task, that is, to incorporate computational and cost factors into the model.

The model for long matrices is mostly new material. It is based on some well known concepts such as expected value, utility function, frequency distribution, setup costs, etc. However the model itself is a new contribution. My contribution here is mostly the building of a new framework, upon which the models previously mentioned are based.

This new framework becomes an enhancement of those models. Their integration into a DSS perspective makes them more useful; since it is now more clear what their role is among the different pieces of software necessary to provide help to the decision maker.

7. Future work

The model we presented in section 5. is still far from being a well tested method. It is mainly a theoretical model. Also, it is not a very sophisticated model from a statistical point of view. There is still plenty of room for improvement in that sense; but I think that the algorithm presented is a good starting point.

The accuracy of the algorithm may be improved if we know, or assume, some specific distribution functions for the attributes. So far we have used only very simple frequency tables in order to calculate $P(XPa_i)$. But if we know, for example, that some of the attributes follow a Poisson distribution or a binomial distribution, etc., then we can calculate the value of $P(XPa_i)$ more accurately.

BIBLIOGRAPHY

Alvarado, R. *El problema del valor multiatributo. Una aproximacion desde la Teoría de Juegos.* Ciencia y Tecnologia _V9, N1, 7-10, 1985.

Alvarado, R. *Un algoritmo para la toma de decisiones en el caso de multiples criterios.* Ciencia y Tecnologia V12, N1, 1988.

Bell, D and Farquhar, P. *Perspectives in utility theory.* Operations Research V34, N1, 179-183, January 1986.

Bui, T. *Building effective multiple criteria decision models: A decision support systems approach.* System. Object. Solut. V4, N1, 3-16, January 1984.

Calpine, H.C. and Golding, A. *Some properties of Pareto Optimal Choices in decisions problems.* Omega. V4, N2, 1412-17, 1976.

Czogala, E. *Multicriteria decision making in terms of probabilistic sets.* Uncertainty and Intelligent Systems, Proceedings. Lecture Notes in Computer Science N313, Springer Verlag, N.Y., 366-372, 1988.

Fishburn, P. C. The Foundations of Expected Utility. Dordrecht, Holland, 1982.

Fishburn, P. C. Utility Theory for Decision Making. Wiley, N. Y., 1970.

Fishburn, P. C. Nonlinear Preference and Utility Theory. John Hopkins U. Pr., Md, 1988

Holsapple C. and Whinston, A. Decision Support Systems: Theory and Applications. Springer Verlag, Berlin, 1987.

Hwang, C. and Yoon, K. Multiatribute Decision Making. Methods and Applications. Lecture Notes in Economics and Mathematical Systems, Springer Verlag, Berlin, 1981.

Karp, R. Upfal, E. and Widgerson, A. *Are search and decision programs computationally equivalent?* Theory of Computing. Proceedings. ACM 17 Symposium, 464-475, May 1985 .

Keeney, L. R. and Raiffa, H. Decisions with Multiple Objectives. Wiley, N.Y. 1976

Kraft, D.H. and Buell, D. *Advances in a Bayesian decision model for user stopping behavior for scanning the output of an information retrieval system.* In Research and Development in Information Retrieval. Edited by C.J. Rijsbergen, pp. 421-431, 1984.

Lazimy, R. *Solving multiple criteria problems by interactive decomposition.* Mathematical Programming. V35, N3, 334-361, July 1986.

Liebowitz, J. *Beyond decision support systems: The role of operations research in expert systems.* Computers and Industrial Engineering. V14, N4 , 415-418, Set. 1988.

Marcotte, O. Soland, R. *An interactive branch and bound algorithm for multiple criteria optimization.* Management Science. V32, N1, 61-75, January 1986

Mitra, G. (Editor). Mathematical Models for Decision Support. Springer-Verlag, Berlin, 1988.

Moore, J.C. and Whinston, A. B. *A model for decision making with sequential information-acquisition.* Part I. Decision Support Systems. V2, N4, 285-307, 1987.

Moore, J.C. and Whinston, A. B. *A model for decision making with sequential information-acquisition.* Part II. Decision Support Systems. V3, N,1, 47-72 , 1987.

Moore, J.C. and Whinston, A. B. *A decision theoretic approach to file search.* Computer Science in Economics and Management V1, N1, 3-20, January 1988.

Raiffa, H. Decision Analysis. Addison Wesley, Reading, Mass., 1970

Reimann, B. and Waren, A. *User oriented criteria for the selection of DSS software.* Communications of ACM, V28, N2, 166-179, Feb. 1985.

Vetschera, R. *An interactive outranking system for multiattribute decision making.* Computers and Operation Research. V15, N4, 311-322, July 1988.

Weber, M. *A method of multiattribute decision making with incomplete information.* Management Science. V31, N22, 1372-1389, Nov. 1985.

Yu, P.L. and Leitmann, G. *Nondominated decisions and cone convextity in dynamic multicriteria decision problems.* Multicriteria Decision Making and Differential Games. Edited by George Leitmann. Plenum Press, N. Y., 1976.

Yu, P.L. Lee, Y. and Stam, A. Multicriteria Decision Making: Concepts, Techniques and Extensions. Plenum Press, N.Y., 1985.

Zionts, S. *Multiple criteria mathematical programming. An overview and several approaches.* Mathematics of Multiple Objective Optimization. Proceedings. Courses and Lecture Notes N289, Springer Verlag, N.Y., 227-273, 1985.