Vetschera, Rudolf

Working Paper

A recursive algorithm for volume-based sensitivity analysis of linear decision models

Diskussionsbeiträge - Serie I, No. 279

Provided in Cooperation with:
Department of Economics, University of Konstanz

Suggested Citation: Vetschera, Rudolf (1996) : A recursive algorithm for volume-based sensitivity analysis of linear decision models, Diskussionsbeiträge - Serie I, No. 279, Universität Konstanz, Fakultät für Wirtschaftswissenschaften und Statistik, Konstanz

This Version is available at:
https://hdl.handle.net/10419/68933

Fakultät für Wirtschaftswissenschaften und Statistik

Rudolf Vetschera

A Recursive Algorithm for Volume-Based Sensitivity Analysis of Linear Decision Models

Diskussionsbeiträge

Postfach 5560
D-78434 Konstanz

Serie I — Nr. 279
Januar 1996

A Recursive Algorithm for Volume-Based Sensitiveity Analyxis of Linear Decision Models

Rudolf Vetschera

Serie I - Nr. 279

Januar 1996

A RECURSIVE ALGORITHM FOR VOLUME-BASED SENSITIVITY ANALYSIS OF LINEAR DECISION MODELS

Rudolf Vetschera

Abstract

One problem impeding the use of linear decision models in practical applications is the difficulty in precisely specifying weights. In this paper, we analyze an approach to establish the stability of a decision, given approximate information on weights. This approach is based on comparing the volumes of regions in weight space, in which different alternatives are optimal. To compute those volumes, a recursive algorithm was developed. The efficiency of that algorithm is analyzed both analytically and via computational experiments.

Zusammenfassung

Grundlage dieser Arbeit bildet ein Konzept für die Sensitivitätsanalyse linearer Entscheidungsmodelle. Die Sensitivität einer Lösung wird dabei durch das Volumen des Bereiches im Parameterraum gemessen, in dem die betrachtete Lösung optimal bleibt. Dieser Ansatz vermeidet einige Probleme üblicher, auf Abstandsgrößen beruhender Sensitivitätsmaße. Die dafür erforderliche Volumensberechnung mehrdimensionaler Polyeder erfordert jedoch hohen Rechenaufwand. Zur Lösung dieses Problems wurde ein rekursiver Algorithmus entwickelt, der in dieser Arbeit vorgestellt und bezüglich seines Laufzeitverhaltens analysiert wird.

1. Introduction

Linear decision models of the form

$$\sum_{k=1}^{K} w_k \cdot a_{ik} \qquad (1)$$

have become a common tool in analyzing decision problems under multiple criteria. Many different approaches using such models have been developed. They differ in the interpretation of the values a_{ik}, which are interpreted for example as partial utility of alternative $\mathbf{a}_i$ in attribute k in multiattribute utility theory (Keeney/Raiffa, 1976), or as local priorities of alternatives in the Analytic Hierarchy Process (Saaty, 1980). Most methods also differ in the way the criteria weights w_k are determined. However, despite all those differences, the approaches using a weighted additive aggregation rule like (1) have one problem in common: methods to measure the w_k's often lead to conflicting results (Schoemaker/Waid, 1982) which sometimes even violate the underlying axioms (Weber et al., 1988; Delquié, 1993; von Nitzsch/Weber, 1993), or are faced with reluctance or resistance by the decision maker to specify precise numerical values (Dickson, 1981; Arbel/Vargas, 1993).

To overcome this problem, several approaches have been developed for dealing with incomplete or imprecise information on criteria weights. Some of these approaches will be described in more detail in the following section. In section three of this paper, we will develop an approach to deal with this problem based on the concept of volume in weight space. This approach allows us to determine the probability that a given alternative will turn out to be optimal, given some (imprecise) a priori information on possible weights. In order to perform the necessary calculations, an algorithm was developed to compute the volume of a polyhedron in n-dimensional space. This algorithm is presented in section four. In section five, the complexity of the algorithm is analyzed both theoretically and via computational experiments. Section six concludes the paper by identifying some areas for future research.

2. Sensitivity Analysis: Previous Approaches

2.1. Overview

Figure 1 provides an overview over different frameworks for sensitivity analysis in linear decision models.

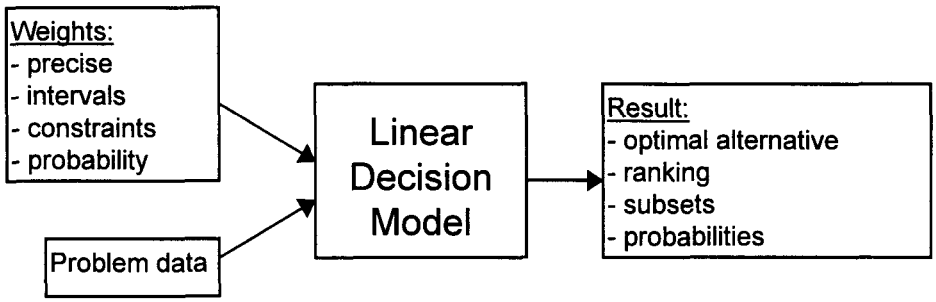

Figure 1: Conceptual Framework

We can classify previous approaches to sensitivity analysis according to three criteria:

1. The inputs to the decision model, i.e. the type of information on weights that is assumed to be available. Apart from the standard case of precise weights, the most common variants are the specification of intervals (i.e. weight w_k must lie between a lower and an upper bound), linear constraints on weights (e.g. $w_3 > 2w_4$ indicating that the weight for attribute 3 must be at least twice that of attribute 4) and probability distributions on weights.

2. The output of the decision model that is studied. In their standard form, linear decision models of the form (1) generate a ranking of alternatives. Approaches for sensitivity analysis are, however, not always concerned with the stability of the entire ranking of alternatives. Some approaches consider for example only conditions under which the optimal alternative remains the same. Other methods are concerned with subsets of the entire ranking, e.g. the first n alternatives, or "large" changes in which an alternative moves by several ranks. Still other approaches deal with the probability of certain changes.

3. The direction in which the analysis is performed. Here we can distinguish between models that generate information about the outputs given some information about the inputs and models which calculate information about the allowable inputs for certain outputs.

In the following subsections we review previous approaches to sensitivity analysis according to the information on weights that is used or generated. In this presentation, we will use the following notation, which has already been introduced in (1):

We consider a decision problem involving K attributes, which are indexed by $k=1,...,K$. Attribute k is given a weight w_k. The decision is to be made among N alternatives, which will be indexed by i,j etc. The evaluation of alternative $\mathbf{a}_i$ in attribute k is denoted by a_{ik}. Vectors of values will be written as boldface letters, e.g. $\mathbf{w}$ is the vector of weights $(w_1,...,w_K)$.

2.2. Interval weights and constraints

Intervals on weights can be interpreted as a special case of constraints and many methods deal with both types of information simultaneously. In this paper, we also will not distinguish further between these two types of information.

This kind of information is used in a wide variety of approaches. The first class of approaches takes a decision-making perspective and tries to generate a (unique) optimal solution given imprecise weight information. This kind of approaches is often called "decision making with incomplete (or partial) information", a review of the relevant literature is given e.g. in (Weber, 1987).

Most techniques of this class are based on linear programming. An alternative $\mathbf{a}_i$ is considered to be potentially optimal, if a weight vector $\mathbf{w}$ can be found which simultaneously satisfies the condition

$$\sum_{k=1}^{K} w_k a_{ik} \geq \sum_{k=1}^{K} w_k a_{jk} \quad \forall j \neq i \tag{2}$$

and the constraints the decision maker has formulated with respect to the w_k's. If, for a certain alternative, the resulting problem is infeasible, it is concluded that this alternatives cannot be optimal given the available information on $\mathbf{w}$. A related concept is that of dominance with respect to a set W of possible weights (Rios Insua/French, 1991; French, 1992): an alternative $\mathbf{a}_i$ dominates another alternative $\mathbf{a}_j$ with respect to a feasible set W of weight vectors, if the evaluation of $\mathbf{a}_i$ is at least as good as that of $\mathbf{a}_j$ for all $\mathbf{w} \in W$ and strictly better for at least one weight vector. For specific types of sets W, (Kirkwood/Sarin, 1985) developed quick tests for this form of dominance.

An approach developed by (Arbel, 1989) considers the entire ranking as output. This approach was developed in the framework of the Analytic Hierarchy Process and is concerned only with one hierarchical level. Therefore, the w_k directly correspond to the evaluation of alternatives in the attribute under considerations. The model developed by Arbel checks whether a consistent ranking of all alternatives can be established for all vectors $\mathbf{w}$ which are feasible under a set of constraints specified by the decision maker.

The two methods described so far take a set of constraints on the weights as given and analyze possible outputs for this preference information. A technique developed by (Hansen et al., 1989) considers the opposite direction: given the optimal solution to a multiobjective linear programming problem, this method determines a range of weights in which the optimal solution remains the same. In the framework of the PROMETHEE outranking method, (Wolters/Mareschal, 1995) formulated a linear programming model to determine the minimum change in weights which would make a different alternative optimal. An analytical solution for a similar problem in the context of additive models is given by (Barron/Schmidt, 1988).

A similar point of view is taken by (Evans, 1984), who develops a measure of the sensitivity of a decision. Evans partitions the set of standardized weight vectors, for which

$$\sum_{k=1}^{K} w_k = 1 \tag{3}$$

holds, into subsets in which a different alternative is optimal. Sensitivity of a certain choice is then defined as the radius of the largest circle that can be inscribed in the subset corresponding to the selected alternative. This approach is extended by (Rios Insua/French, 1991), who considered the ratio of the distance of a given weight vector at which the optimal alternative changes and the largest possible change in weights as a measure of sensitivity.

2.3. Probabilities

Only few approaches have been presented which explicitly take probabilities into account. One approach was developed by (Saaty/Vargas, 1987), who used a simulation model, in which samples are draw from judgment intervals in comparison matrices of the AHP. The model then proceeds to compute intervals for the evaluation of alternatives and finally probabilities for rank reversals between alternatives (Arbel/Vargas, 1993).

3. The Volume-Based Approach

The measures of sensitivity developed so far and reviewed in the preceding section measure sensitivity mostly in terms of distances from a certain starting point **w** to a point where some change in the output occurs. Using distances for this purpose has two disadvantages: The first problem is the selection of a distance measure. It has been shown that the use of different distance measures like various ℓ_p norms or the Tchebycheff norm will lead to different results (Rios Insua/French, 1991). The second problem is that distances are not preserved by projections. If, for example, substitutions are made based on a scaling condition of weights, the resulting distortion must explicitly be taken into account in performing distance calculations (Schneller/Sphicas, 1985).

We therefore propose to use volumes instead of distances to measure sensitivity. The comparison of volumes was in a way proposed by (Rios Insua/French, 1991), who viewed their ratio of radii of circles as an approximation to the ratio of areas, but did not extend this thought further.

The idea of using volumes was also used in the context of decisions under risk by (Starr, 1962), who developed a "domain criterion" for ranking alternatives. His "domain" corresponds to the volume of a region in probability space in which an alternative is optimal. This approach was developed further by (Schneller/Sphicas, 1983) and (Eiselt/Langley, 1990). Its applicability to the context of multi-criteria problems was first suggested by (Charnetski/Soland, 1978), who used Monte Carlo simulation for approximately determining the volumes. The concept was also discussed by (Erkut/Tarimcilar, 1991)

and (Eiselt/Laporte, 1992). These authors, however, did not develop a practical algorithm for actually computing multidimensional volumes. (Antunes/Clímaco, 1993) used areas in a graphical method to analyze problems with three attributes, but this approach cannot be extended to problems with more attributes.

Our approach can be outlined as follows: given some a priori information on weights in the form $\mathbf{w} \in W$, we partition the set W into regions in which different alternatives are optimal. For a given vector of weights $\mathbf{w}$ and its associated optimal alternative $\mathbf{a}^*$, the sensitivity of that decision is then defined as the ratio of the volume of the partition associated with $\mathbf{a}^*$ to the entire volume of W. This ratio has a straightforward interpretation: assuming that weight vectors are uniformly distributed over the set W, it corresponds to the probability that $\mathbf{a}^*$ is the optimal alternative if a weight vector is randomly selected from W.

To formalize the concept, we will first consider the case in which no a priori information is available other than the usual scaling of weights to

$$\sum_{k=1}^{K} w_k = 1 \tag{4}$$

In order to be the optimal alternative, the evaluation of an alternative $\mathbf{a}_i$ according to (1) has to be better than the evaluation of all other alternatives. We therefore obtain the following linear model, which describes the set of weights for which alternative $\mathbf{a}_i$ is optimal:

$$\begin{aligned} &\sum_{k=1}^{K} w_k a_{ik} \geq \sum_{k=1}^{K} w_k a_{jk} \quad \forall j \neq i \\ &\sum_{k=1}^{K} w_k = 1 \end{aligned} \tag{5}$$

The last equation in (5) can be used to substitute for one weight, e.g. w_K. It should be noted that in contrast to distance-based measures, the ratio of volumes will not be affected by this transformation (Schneller/Sphicas, 1983). We obtain the following set of inequalities:

$$\begin{aligned} &\sum_{k=1}^{K-1} \Big((a_{jk} - a_{ik}) - (a_{jK} - a_{iK}) \Big) \cdot w_k \leq a_{iK} - a_{jK} \quad \forall j \neq i \\ &\sum_{k=1}^{K-1} w_k \leq 1 \end{aligned} \tag{6}$$

(6) defines a polytope in K-1 - dimensional space. We will call this polytope $\mathbf{a}_i$'s region of optimality. Consider, for example, the case K=3. Then (6) defines a polygon, which is part of the unit triangle in w_1/w_2 space. Taken together, conditions (6) for all alternatives partition that triangle into different regions (figure 2).

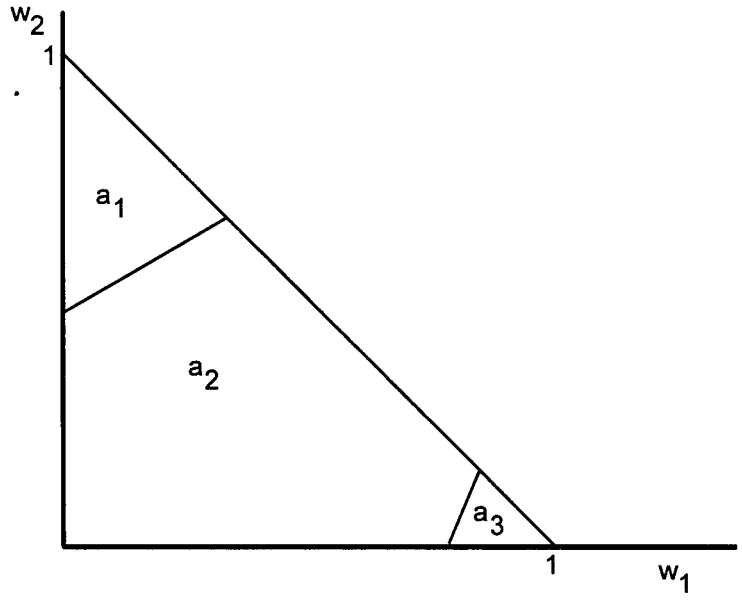

Figure 2: Partitioning of the unit triangle

In figure 2, the unit triangle is partitioned into three regions, in which alternatives $\mathbf{a}_1$, $\mathbf{a}_2$ and $\mathbf{a}_3$, respectively, are optimal. The region of optimality for $\mathbf{a}_3$ is very small compared to the region for $\mathbf{a}_2$. Given no additional a priori information on the weights, $\mathbf{a}_2$ is therefore the more likely choice and conversely, a decision to select $\mathbf{a}_3$ should be analyzed more carefully.

4. The Algorithm

In this section, we will develop an algorithm for computing the volume of the K-1 - dimensional polytope defined by (6). We will first introduce the main ideas graphically for the 2-dimensional case and then develop the algorithm analytically.

4.1. Graphical exposition

Figure 3 graphically explains the calculation of the area of a polygon in two-dimensional space:

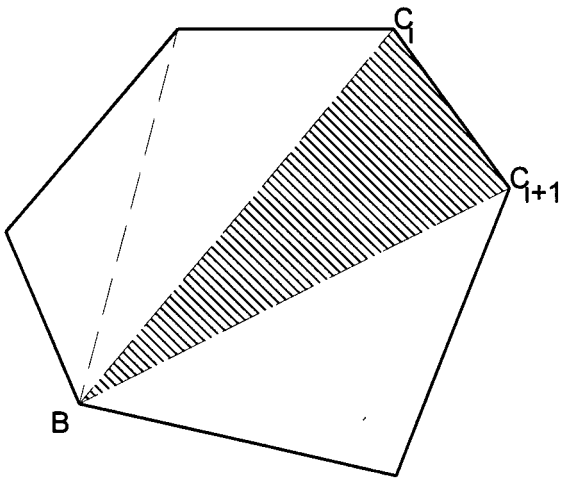

Figure 3: Computing the area of a polygon

Starting from an arbitrary corner B, we partition the polygon into triangles by connecting B to all other corners, e.g. C_i and C_{i+1}. The area of each triangle between B and two other points C_i and C_{i+1}, which are adjacent to each other, can then be calculated as the determinant of the matrix formed by the vectors (C_i - B) and (C_{i+1} - B). By summing the areas of all these triangles, we obtain the total area of the polygon.

The same idea can be applied to calculate the volume of a 3-dimensional polyhedron: again we start at an arbitrary corner point B_0 of the polyhedron. Using the same method as above, we partition each facet of the polyhedron into triangles emanating from a starting point B_i on the facet. For two neighboring points C_j and C_{j+1} on the facet, we can compute the volume of the tetrahedron (B_0,B_i, C_j, C_{j+1}). The total volume of the polyhedron is obtained by adding up the volumes of all such tetrahedra.

4.2. The recursive procedure

To develop the algorithm more formally, we first rewrite (6) as a system of linear equations by introducing slack variables s_j. Without loss of generality, we also assume that the volume of polyhedron (6) is to be computed for alternative N, so the first $N-1$ inequalities in (6) refer to alternatives $\mathbf{a}_1$ to $\mathbf{a}_{N-1}$. We obtain:

$$\begin{aligned} &\sum_{k=1}^{K-1}\Big((a_{jk}-a_{ik})-(a_{jK}-a_{iK})\Big)\cdot w_k + s_k = a_{iK}-a_{jK} \quad \forall j \neq i \\ &\sum_{k=1}^{K-1} w_k \leq 1 \end{aligned} \tag{7}$$

This is a system of N linear equations in $N+K-1$ variables. A corner of the polytope corresponds to a basis of (7), which contains N basic and $K-1$ nonbasic variables. The number of nonbasic variables is equal to the dimension of the problem.

To formalize the ideas we have developed graphically, we introduce the concept of a q-dimensional facet. A q-dimensional facet of an m-dimensional problem is a set of basic solutions in which q nonbasic variables can be exchanged for other variables and m-q nonbasic variables must remain nonbasic. For example, a one dimensional facet is a line segment linking two adjacent basic solutions. In moving along that line, only one nonbasic variable is replaced, the other m-1 nonbasic variables remain the same. As another example, consider a two-dimensional facet of a three-dimensional problem (m=3, q=2): that facet is defined by m-q = 1 constraint, so the slack variable of that constraint must remain nonbasic, while the other q = 2 nonbasic variables can be replaced by other variables.

In order to compute the volume of the $K-1$- dimensional polyhedron defined by (7), we have to analyze each of its $K-2$ - dimensional facets, which in turn contains $K-3$ - dimensional facets and so

on until we end up with line segments which finally define the *K-1*- dimensional tetrahedra of which the volume can be computed.

A similar decomposition approach was developed by (Cohen/Hickey, 1979). Their approach, however, assumes that the polyhedron is given in the form of a set of vertices, from which the facets must be constructed. In the problem we consider here, those facets are already implicitly given via the constraints. To apply the algorithm of (Cohen/Hickey, 1979), one would first have to compute and store all vertices, from which the facets would be reconstructed. Apart from the storage requirement associated with storing all vertices, this approach is also clearly inefficient from a computational point of view, since information on the facet structure, which is readily available from the constraints, has to be reconstructed from the vertices.

The problem of finding all *K-2* - dimensional facets to a *K-1* - dimensional problem is closely related to the problem of finding all vertices of a polyhedron, for which several algorithms have been developed (Matheiss/Rubin, 1980). The algorithm developed here is a recursive variant of the algorithm by (Manas/Nedoma, 1968), which basically consists of a branching procedure which examines all neighboring basic solutions to a given basis. For each basis, we then proceed by analyzing its adjacent facets of next lower dimension, which have not yet been analyzed.

To provide a compact representation of the algorithm, we use the following notation: let `P` be a complete description of a basic solution and its associated simplex tableau. By `P.B`, we denote the set of basic variables, `P.NB` is the set of nonbasic variables. `P.w` is the set of weights associated with the current solution. The algorithm uses a procedure `pivot(in,out,Pold,Pnew)`, which performs one pivot step on problem `Pold`. The variable entering the basis is denoted by `in`, the variable leaving the basis by `out` and `Pnew` is the new state of the problem after the pivot step. A global variable `w0` represents the coordinates of an initial solution. The algorithm can then be formulated as a recursive procedure `Visit`, which has the following parameters:

Parameter	Description
`d`	Dimension of the current subproblem
`u`	A set of variables which must remain nonbasic
`m`	A matrix, describing some points of the current K-1-dimensional tetrahedron
`v`	The set of basic solutions (of dimension `d`) which already has been visited
`f`	The set of `d-1` - dimensional facets already visited for the current subproblem of dimension `d`
`P`	The current state of the simplex tableau
`Vol`	A variable in which the volume is being built up

```
procedure Visit(d, u, m, v, f, P, vol);
variables
v1, f1, facet: set
i: integer
begin
  v := v ∪ {P.B}
  if d = 1 then
    vol := vol + DET(m)
  else
    for i ∈ P.NB\u do
      facet := u ∪ {i}
      if facet ∉ f then
        append (P.w - w0) as last column to m
        f1:={}
        v1:={}
        Visit(d-1, facet, m, v1, f1, vol)
        f:=f ∪ {facet}
        end if
      end for
    for i∈P.NB\u do
      Pivot(i, out, P, P1)
      if P1.B ∉ v then
        Visit(d,u,m,v,f,P1,vol)
        end if
      end for
    end if
end visit
```

The procedure twice iterates across all nonbasic variables which are not in set u. These two loops could be combined into one, but the algorithm can better be explained by considering them separately. In the first loop, the d-1-dimensional subproblems of P are generated. As we have already explained, a q-dimensional facet can be described by m-q nonbasic variables which must remain nonbasic. These are stored in the variable facet, which thus uniquely identifies the facet under consideration. For each facet, a new subproblem of dimension d-1 is solved. It should be noted that new sets v1 and f1 are created for each subproblem to be solved. Completion of that subproblem is then recorded in set f.

Once the recursive process has reached dimension d=1, a complete K-1-dimensional tetrahedron has been constructed and its volume can be computed as the determinant of the vectors forming that tetrahedron. Since the recursion on subproblems has gone through K-1 levels, matrix m is square.

The second loop generates subsequent problems at the same dimension. This process is a recursive formulation of the algorithm by (Manas/Nedoma, 1968) and simply consists of branching on all nonbasic variables and visiting all those among the resulting basic solutions which have not yet been visited.

The process is initiated at some feasible basis P. The weights associated with P are the "top" of all the tetrahedrons that will be calculated. This point is connected to all the points generated by forming the vectors (P.w - w0). The initial dimension of the problem is *K-1*.

The algorithm can be easily adapted to problems in which the weights are restricted to a certain subset W of the K-1-dimensional simplex as long as those restrictions are linear. If restrictions on weights are given in the form of linear inequalities, these can simply be added to (7) with their respective slack variables and the same algorithm can be applied. The number of nonbasic variables and therefore the dimension of the problem remains unchanged in this case. If restrictions on weights are formulated as equations, they reduce the dimension of the problem. This kind of restrictions can be dealt with by substituting for one weight and solving the reduced problem.

4.3. Implementation aspects

As we will show in the next section, the algorithm introduced above requires considerable computational effort. In implementing the programs used to generate the computational results presented in the next section, several techniques have been developed to reduce the computational requirements. These methods do not alter the basic structure of the algorithm as presented. Still, by introducing these techniques, the running time was reduced by a factor of about 10.

4.3.1. Pre-calculation of bases

Since procedure `Visit` operates on facets of different dimensions, each vertex of the original polyhedron will be visited several times. Computation time can thus be reduced if the basic solutions to the problem are computed once, before procedure `Visit` is started, and are not re-computed in each recursive invocation of the procedure.

The ratio between the total number of invocations of procedure `Visit` and the number of vertices of the polyhedron depends on the dimension d of the problem and can be described by a function f, which is defined recursively as

$$\begin{aligned} f(0) &= 0 \\ f(d) &= f(d-1)\cdot d + 1 \end{aligned} \tag{8}$$

This function thus increases faster than the factorial. Especially for larger dimensions, the saving in computing time achievable by pre-calculation of bases is therefore substantial. On the other hand, complex problems might posses a large number of basic solutions, so the amount of information to be saved for each basis is important.

It is not necessary to save the entire simplex tableau for each basis. On the other hand, unlike the algorithm of (Cohen/Hickey, 1979), the present algorithm utilizes some information from the simplex tableau, so it is not sufficient to store just the coordinates of the vertices. In order to apply procedure `Visit`, a graph structure is constructed, where each node of the graph represents a vertex of the polyhedron. In each vertex, the following information is stored:

1. The coordinates of the vertex
2. The set of basic variables
3. The set of nonbasic variables
4. For each nonbasic variable, a pointer to the basis resulting from replacement of that variable as well as the index of the replaced basic variable is also stored.

The pointers to adjacent basic solutions correspond to the arcs of the graph. It can easily be seen from the algorithm that this information is sufficient for execution of the procedure `Visit`.

4.3.2.Avoiding empty tetrahedra

Since it processes each facet at each dimension separately, procedure `Visit` generates tetrahedra of volume zero. A tetrahedron will be empty if any point added to matrix `m` at some higher dimension is re-entered into the matrix at a lower dimension or there is a linear dependency among the points in the matrix. The first case can easily be checked and can lead to the elimination of an entire subtree of recursive invocations, if such a duplication is detected at a high dimension. The second condition can only be detected by computing the determinant, which is necessary for computing the volume, too and therefore cannot be used to reduce computing time. Nevertheless, by checking for duplication of vectors in `m`, the number of empty tetrahedra was reduced from over 90% to about 50% of all the tetrahedra generated in the computational experiments.

5. Complexity and Computational Results

The algorithm developed above is highly recursive. It can therefore be suspected that it will lead to considerable computation times. In this section, we will analyze how different parameters of the problem will affect computation times, first by theoretical considerations and then by presenting the results of computational experiments. Specifically, we will consider the number of alternatives (N) and the number of attributes (K).

5.1. Theoretical analysis

5.1.1.Change in the number of alternatives

Two different cases have to be considered when analyzing the effects of an increase of the number of alternatives on the computational effort involved. One has to distinguish between a situation in which the volume of the region of optimality for one alternative is to be computed and a situation in which one wants to analyze a complete partition of set W. If the algorithm is used for sensitivity analysis with regard to an existing solution, the first situation applies, while other uses might lead to the second situation.

When we consider only the region of optimality for one alternative, the effects of increasing the number of alternatives will not be dramatic. In the first loop of the algorithm, each recursive invocation of the procedure will decrease variable d, so the maximum level of recursion in this loop depends only on the number of attributes, not on the number of alternatives.

The second loop generates subproblems of the same dimension. The number of problems generated through this recursion is equal to the number of vertices of the polyhedron. From the Upper Bound Conjecture (Matheiss/Rubin, 1980) we obtain the following upper bound on that number:

$$\binom{N-[K/2]}{N+1-K}+\binom{N-[(K+1)/2]}{N+1-K} \tag{9}$$

where [.] denotes rounding down to the nearest integer. Increasing N in (9) will increase computational effort linearly. We first consider the case of K being even. The Upper Bound Conjecture then becomes:

$$\binom{N-K/2}{N+1-K}+\binom{N-K/2}{N+1-K} \tag{10}$$

so the two terms are identical. From the definition of binomial coefficients, we can obtain the factor by which (10) increases when N is replaced by $N+1$ as:

$$\binom{N+1-K/2}{N+2-K}\Big/\binom{N-K/2}{N+1-K}=\frac{N+1-K/2}{N+2-K} \tag{11}$$

which can be rewritten as:

$$\frac{N+1-K/2}{N+2-K}=1+\frac{K/2-1}{N+2-K} \tag{12}$$

Since the second term in (12), and a similar expression which can be derived for odd K, decreases hyperbolically in N, the number of vertices and therefore the number of subproblems to be solved in the second loop of our algorithm will increase only linearly with the number of alternatives. However, this increase of vertices takes place at all levels of recursion from the first loop: not only the number of vertices of the K-1-dimensional polyhedron increases but also the number of vertices of the K-2, K-3,... -dimensional polyhedra increases. Furthermore, the computational effort for each pivot step increases linearly in the number of rows of the simplex tableau, which is equal to N. Therefore, the total computation time for the algorithm could increase by a polynomial factor with an exponent equal to the number of attributes.

Practically, this increase will probably only be about linear, since simulation studies like (Matheiss/Rubin, 1980) have shown that for realistic problems, the number of vertices increases much

slower than the Upper Bound Conjecture suggests. This result has also been conformed by our experiments presented in the following subsection.

To obtain a complete partition of set W, one has to solve problem (6) for each alternative. However, this does not lead to a linear increase in computation time, since for some alternatives no set of weights exists which would make that alternative optimal. This means that the region of optimality for such an alternative is empty. This condition can easily be verified, if problem (6) does not possess a feasible solution. We therefore can expect the effect on computation time due to this factor to be less than linear.

5.1.2. Change in the number of attributes

For changes in the number of attributes, however, we have to expect a rapid growth of computing time. For any facet generated in the first loop, a complete `d-1` - dimensional subproblem has to be solved. Increasing the number of attributes by one therefore increases the computational effort by a factor which is equal to the number of facets in the upper problem. Since this factor is again bounded by the Upper Bound Conjecture (9), we have in the worst case to expect faster than exponential growth in computation times. Even if the number of facets increases only linearly in the number of dimensions, growth in overall computation time would be faster than exponentially.

5.2. Computational results

To verify the theoretical considerations of the above section, several computational experiments were performed. In this experiments, the algorithm developed in this paper was tested against a simple simulation approach, in which the optimal alternative was computed at equally spaced grid points. The volume of the region of optimality for each alternative was then computed as the number of grid points in which that alternative was optimal, divided by the total number of feasible grid points.

Since that simulation approach generates the regions of optimality simultaneously for all alternatives, in the first set of experiments the recursive algorithm was also run for all alternatives. Table 1 lists the average CPU time (in seconds) and their standard deviations for 10 experiments each for various numbers of attributes and alternatives. All tests were run on the Silicon Graphics Power Challenge at the University of Konstanz computing center using the Irix Pascal compiler.

	Attributes			
Alternatives	3	4	5	6
5	0.016 0.005	0.069 0.019	0.790 0.350	21.627 11.485
10	0.063 0.007	0.192 0.045	2.011 0.826	73.080 44.708
15	0.179 0.016	0.427 0.111	3.683 1.120	90.160 45.187
20	0.443 0.016	0.833 0.126	4.798 1.754	155.378 76.086

Table 1: CPU times (mean and standard deviation for 10 experiments) for evaluating all alternatives

Figure 4 shows the development of running times for changing numbers of alternatives. Since running times increase sharply with the number of attributes, these results are normalized so that the running time for the case of five alternatives is set to 1 for all numbers of attributes.

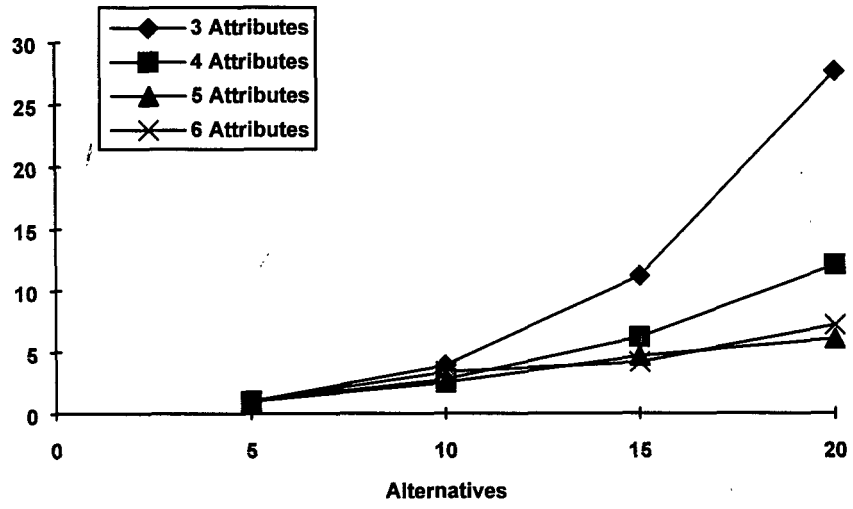

Figure 4: Relative running times with increasing number of alternatives

The figure shows that, for a larger number of attributes, an increase in the number of alternatives leads to an approximately linear increase in the calculation time. For problems with few attributes, the increasing number of vertices to be generated in the initial phase leads to an exponential increase.

In the next figure, we analyze the effect of changes in the number of attributes. The theoretical considerations in the preceding subsection already have led to the conclusion that increasing the number of attributes will increase the running time exponentially or even faster. This effect is clearly visible from the results.

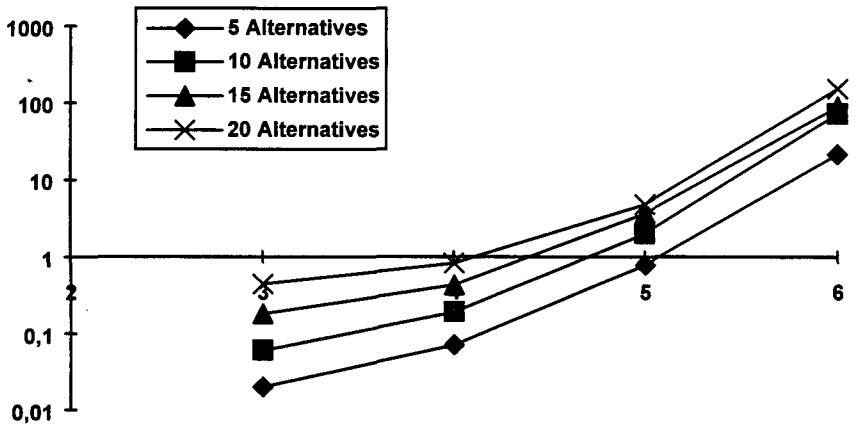

Figure 5: Running times and number of attributes

To evaluate the performance of the algorithm, it was then compared to the running time of the naive simulation approach. The running time of the naive simulation approach depends on the step size of grid points. Instead of choosing an arbitrary step size, the two algorithms were compared by computing the number of steps in the naive simulation approach which would lead to (approximately) the same running time as the recursive algorithm. This also provides an estimate of the precision of the naive simulation approach which would be possible using the same resources as the recursive algorithm.

The equivalent number of steps in the naive approach was calculated as follows: In the naive approach, the unit interval was divided into n steps for each attribute. Since the weights sum up to one, only weights for K-1 attributes were created in this way, from which the weight for the last attribute can be directly computed. Denoting the time needed for processing one grid point by α, the total computing time T_s for the naive simulation approach for K attributes is

$$T_S = \alpha \cdot n^{K-1} \tag{13}$$

First, the naive simulation approach was performed for some arbitrary number of steps n, leading to a time T_s. From the actual computation time T_R of the recursive algorithm, the equivalent number of steps for the naive simulation approach n_e was then calculated as

$$n_e = n \cdot \left(\frac{T_R}{T_S} \right)^{1/K-1} \tag{14}$$

Experiments indicated that using this approach, the running time of the recursive algorithm could indeed be approximated by the naive simulation approach with a precision of about 10%.

Table 2 lists the means and standard deviations for n_e for the experiments carried out. Lower numbers here indicate better performance of the recursive algorithm, since a lower number of steps in the naive simulation approach will lead to less precise results.

	Attributes			
Alternatives	3	4	5	6
5	52.0	20.8	16.7	17.7
	16.53	2.15	2.58	1.95
10	83.3	25.2	19.0	20.2
	10.22	2.25	2.49	3.16
15	113.6	29.7	20.5	20.2
	9.56	2.31	1.72	2.3
20	153.4	34.3	20.4	21.4
	8.04	1.83	2.12	2.22

Table 2: Equivalent number of simulation steps (all alternatives)

The table indicates that the recursive algorithm initially gains performance relative to the naive simulation approach when the number of attributes increases, and then the running time of both algorithms evolves in parallel. With respect to the number of alternatives, the recursive algorithm looses performance when the number of alternatives increases. This result is not surprising, since in the naive simulation approach, additional alternatives only require that the score of the alternatives be calculated at each grid point, while the effort associated with generating the grid points remains the same.

In another set of experiments, the algorithm was run only for one alternative. This setting corresponds more closely to the case of sensitivity analysis, where one starts from a given optimal solution. Table 3 presents the resulting CPU times (average and standard deviations) for 10 experiments in every parameter combination. Since total times were less in these experiments, it was also possible to run experiments for seven attributes.

	Attributes				
Alternatives	3	4	5	6	7
5	0.006	0.024	0.243	6.513	279.343
	0.005	0.005	0.082	2.222	44.206
10	0.011	0.044	0.439	15.241	884.912
	0.003	0.007	0.085	6.086	251.099
15	0.023	0.058	0.661	19.266	1,384.114
	0.005	0.011	0.084	5.617	337.816
20	0.038	0.094	0.750	25.252	2,107.716
	0.004	0.018	0.250	7.889	647.318

Table 3: Running times (mean and standard deviation) for single alternative

Figures 6 and 7 provide the equivalent information to figures 4 and 5 for the case of single alternatives. The results obtained here correspond very closely to those of the previous case.

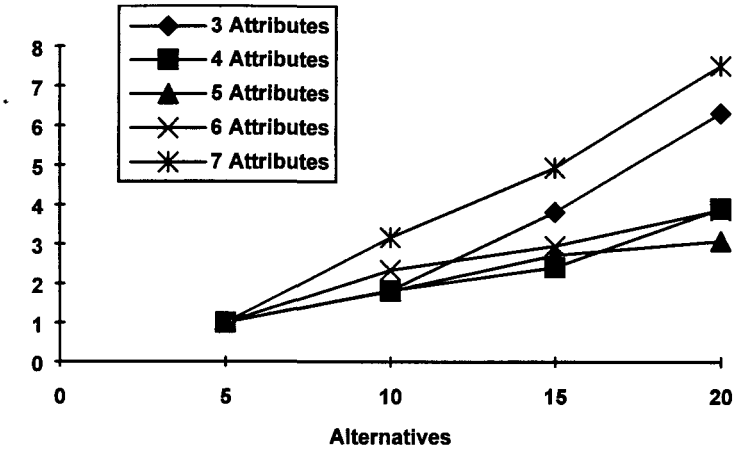

Figure 6: Relative running times for single alternatives when increasing the total number of alternatives

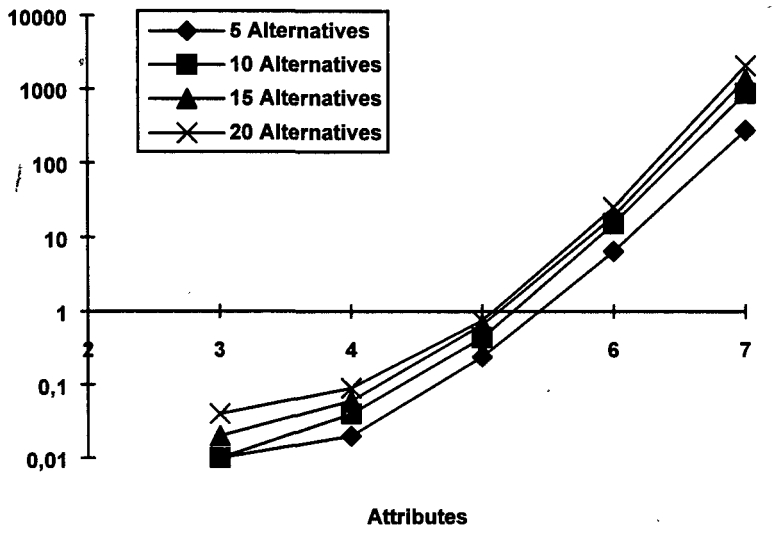

Figure 7: Computation time for increasing number of attributes - single alternative

Here, the computing time increases almost linearly with the number of alternatives for all numbers of attributes, although the variation is greater. Additional attributes cause the calculation time to increase again slightly faster than exponentially.

6. Conclusions and Topics for Further Research

In this paper, we have analyzed the concept of volume-based sensitivity analysis for linear decision models and presented a recursive algorithm by which the necessary calculations can be performed.

The volume-based approach presented here has several applications. As already discussed, it can be used as a tool for sensitivity analysis. The volume of one alternative's region of optimality corresponds to the probability that a decision maker using random weights will select that alternative. Therefore, one can interpret a low value of that volume as an indication that the decision made is

rather specific for the actual set of weights used and thus for the actual decision maker involved. This interpretation can easily be conveyed to the user.

The approach developed here can not only be applied to determine the region of optimality for a single alternative. In the same way, regions in weight space could be analyzed in which other conditions hold. For example, by a simple reformulation of model (6), one can also determine the volume of the region in which the entire ranking of alternatives remains unchanged.

As a tool for sensitivity analysis, the volume-based approach offers several advantages over other approaches. In contrast to distance-based approaches, it does not require the selection of a distance measure, which has been shown to be a crucial factor in distance-based approaches. It is also not sensitive to distortions due to projection into K-1-dimensional space, as are distance-based approaches (Schneller/Sphicas, 1985).

The volume-based approach can also be used in other fields. It generates a probability distribution over alternatives selected given partial (or even no) information on the decision maker's preferences. One can therefore also think of this approach as a tool for modeling an unknown (or partially known) decision maker's preferences. Such a tool could be used, for example, in an agency setting to model preference of an agent about whom the principal has only partial information.

The largest problem which impedes the use of this approach is the computational effort required by the recursive algorithm. It should be noted, however, that the implementation used for the computational tests was designed for easy modification and testing. With sufficient tuning of the implementation, problems with seven attributes could probably be solved in more reasonable time. This might be sufficient for practical purposes, since seven attributes are often regarded as a limit on the cognitive abilities of decision makers, which should not be exceeded, at least not on one level of a hierarchy of attributes (Miller, 1956).

Another approach, which could considerably speed up computation, is parallelization of the algorithm. The recursive algorithm introduced here is extremely well suited for massive parallel systems. The subproblems of lower dimension, which are generated during the first loop of the algorithm, are completely independent of each other and do not interact with the upper level once they are generated. They therefore could be dispatched to different processors without problem, leading to a drastic reduction in overall execution time of the algorithm.

References

Antunes, C.H.; Clímaco, J. (1993): *Decision aid for discrete alternative multiple criteria problems: A visual interactive approach.* Information and Decision Technologies 19: 185-193.

Arbel, A. (1989): *Approximate articulation of preference and priority derivation.* European Journal of Operational Research 43: 317-326.

Arbel, A.; Vargas, L.G. (1993): *Preference simulation and preference programming: robustness issues in priority derivation.* European Journal of Operational Research 69: 200-209.

Barron, H.; Schmidt, C.P. (1988): *Sensitivity Analysis of Additive Multiattribute Value Models.* Operations Research 36: 122-127.

Charnetski, J.R.; Soland, R.M. (1978): *Multiple-Attribute Decision Making With Partial Information: The Comparative Hypervolume Criterion.* Naval Research Logistics Quarterly 25: 279-288.

Cohen, J.; Hickey, T. (1979): *Two Algorithms for Determining Volumes of Convex Polyhedra.* Journal of the ACM 26: 401-414.

Delquié, P. (1993): *Inconsistent Trade-Offs Between Attributes: New Evidence in Preference Assessment Biases.* Management Science 39: 1382-1395.

Dickson, G.C.A. (1981): *An Empirical Examination of the Willingness of Managers to Use Utility Theory.* Journal of Management Studies 18: 423-434.

Eiselt, H.A.; Langley, A. (1990): *Some Extensions of Domain Criteria in Decision Making under Uncertainty.* Decision Sciences 21: 138-153.

Eiselt, H.A.; Laporte, G. (1992): *The use of domains in multicriteria decision making.* European Journal of Operational Research 61: 292-298.

Erkut, E.; Tarimcilar, M. (1991): *On Sensitivity Analysis in the Analytic Hierarchy Process.* IMA Journal of Mathematics Applied in Business & Industry 3: 61-83.

Evans, J.R. (1984): *Sensitivity Analysis in Decision Theory.* Decision Sciences 15: 239-247.

French, S. (1992): *Mathematical programming approaches to sensitivity calculations in decision analysis.* Journal of the Operational Research Society 43: 813-819.

Hansen, P.; Labbé, M.; Wendell, R.E. (1989): *Sensitivity analysis in multiple objective linear programming: The tolerance approach.* European Journal of Operational Research 38: 63-69.

Keeney, R.L.; Raiffa, H. (1976): *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* J. Wiley & Sons, New York.

Kirkwood, C.W.; Sarin, R.K. (1985): *Ranking with Partial Information: A Method and an Application.* Operations Research 33: 38-48.

Manas, M.; Nedoma, J. (1968): *Finding All Vertices of a Convex Polyhedron.* Numerische Mathematik 12: 226-229.

Matheiss, T.H.; Rubin, D.S. (1980): *A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets.* Mathematics of Operations Research 5: 167-185.

Miller, G.A. (1956): *The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.* Psychological Review 63: 81-97.

Rios Insua, D.; French, S. (1991): *A framework for sensitivity analysis in discrete multi-objective decision-making.* European Journal of Operational Research 54: 176-190.

Saaty, T.L. (1980): *The Analytic Hierarchy Process.* McGraw-Hill, New York.

Saaty, T.L.; Vargas, L.G. (1987): *Uncertainty and rank order in the Analytic Hierarchy Process.* European Journal of Operational Research 32: 107-117.

Schneller, G.O.; Sphicas, G.P. (1983): *Decision Making Under Uncertainty: Starr's Domain Criterion.* Theory and Decision 15: 321-336.

Schneller, G.O.; Sphicas, G.P. (1985): *On Sensitivity Analysis in Decision Theory.* Decision Sciences 16: 399-409.

Schoemaker, P.J.H.; Waid, C.C. (1982): *An Experimental Comparison of Different Approaches to Determining Weights in Additive Utility Models.* Management Science 28: 182-196.

Starr, M.K. (1962): *Product Design and Decision Theory.* Prentice Hall, Englewood Cliffs.

von Nitzsch, R.; Weber, M. (1993): *The Effect of Attribute Ranges on Weights in Multiattribute Utility Measurements.* Management Science 39: 937-943.

Weber, M. (1987): *Decision making with incomplete information.* European Journal of Operational Research 28: 44-57.

Weber, M.; Eisenführ, F.; von Winterfeldt, D. (1988): *The Effects of Splitting Attributes on Weights in Multiattribute Utility Measurement.* Management Science 34: 431-445.

Wolters, W.T.M.; Mareschal, B. (1995): *Novel types of sensitivity analysis for additive MCDM methods.* European Journal of Operational Research 81: 281-290.