A bisection method for generating random utility functions in SMAA Extended Abstract

Rudolf Vetschera[1] and **Luis Dias**[2]

1 Introduction

In many applications of decision analysis, alternatives and their performance levels in different attributes are known, but the subjective evaluation of performance remains vague. In utility-based methods, the subjective evaluation of performance levels is usually represented by a marginal utility function defined for each attribute.

One popular method to deal with such vague and incomplete information about preferences is the Stochastic Multiattribute Acceptability Analysis (SMAA) originally developed by Lahdelma et al. [2]. SMAA has evolved into a widely popular method to deal with incomplete information on preferences [4]. It uses Monte-Carlo methods to sample from a space of possible preference parameters, and uses the output of the decision model for these parameters to derive stochastic information about rankings of alternatives and preference relations. From this stochastic information, crisp rankings can be obtained by a variety of methods [6]. The SMAA approach can also be fruitfully combined with preference learning [1].

Originally, incomplete information on preferences considered in SMAA referred to uncertainty about attribute weights. For this problem, specific methods were developed such as the hit-and-run method, that allows for an efficient sampling of weight vectors in a restricted search space [5]. Later on, SMAA was extended to situations in which performance levels of alternatives are only available in the form of rankings [3] and when performance levels are known, but the associated utility values are not [1]. In this work, we address the latter case. We develop methods to randomly generate (marginal) utility functions for a set of given performance levels in each attribute. We consider that the utility functions should be monotonically increasing (but the methods can be easily adapted to the case of decreasing functions). Typically, these performance levels will correspond to the performance levels of existing alternatives in each attribute. While there are specific methods like hit-and-run for generating random weights, the problem of generating utility values has received less attention in literature.

Although it has received less attention in literature, this problem is not trivial. A straightforward approach of randomly generating arbitrary values, sorting them and utilizing them as utility scores of the corresponding performance levels can cause a bias in the shape of the resulting utility function if the distribution of performance levels is skewed. Consider the case that the distribution of performance levels of alternatives is skewed to the right. This is quite likely. If decision makers seek for or generate alternatives, they will try to find alternatives which offer good performance in a majority of attributes (and possibly weak performance in the remaining few attributes). Thus the total set of performance levels will contain more good than weak levels in each attribute. If utility values are drawn from a uniform distribution, the expected difference between successive utility levels is equal, while the expected difference between performance levels decreases for higher performance levels. Taken together, these two effects imply that the slope of the generated utility function increases in performance, i.e. the generated utility function has on average a convex shape.

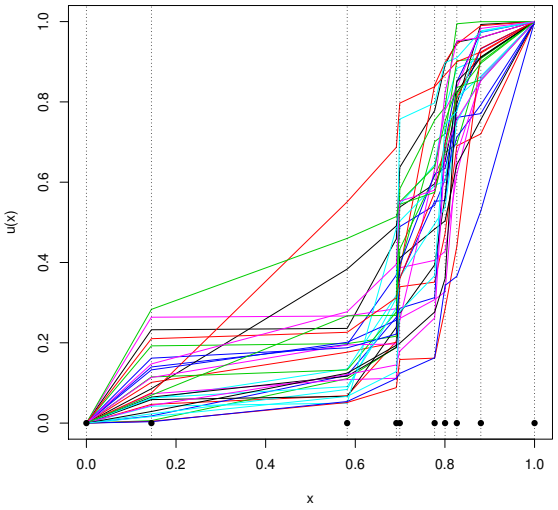

Figure 1. Utility functions generated by straightforward approach

This effect is illustrated in Figure 1. Here we created 30 utility functions by assigning random values to 10 performance levels (shown as dots) drawn from a right-skewed distribution. To generate such a skewed distribution, performance levels were drawn from a density function that increases li-

[1] University of Vienna, Austria
[2] University of Coimbra, Portugal

nearly from zero to two in the $(0, 1)$ interval. Obviously, most of the generated utility functions are biased towards a convex shape.

The bias illustrated in Figure 1 would not occur if all the performance levels were equally spaced, as is often assumed when piece-wise utility functions are used as approximations. However, Figure 1 suggests another concern, which is also present even when all the performance levels are equally spaced: the utility functions generated do not appear to be very different, i.e. variability is low (this becomes worrisome if we generate utility values for more than a just a few different performance levels).

2 A bisection method

If a utility function is approximated by only three points (utility values at the endpoints of its domain and in the mid of the domain), the utility value of the midpoint determines whether the utility function is concave or convex. The function is concave if the utility of the midpoint is larger than the average of the utility values of the endpoints, and convex otherwise. The same argument can be applied to any interval within the domain of the utility function. We therefore propose to generate utility values by successively considering smaller and smaller intervals in the domain, starting with a split of the entire domain, then considering the upper and lower part of the domain and so on, until the desired level of resolution is reached. This approach is related to the bisection method of finding zeros of functions, so we label it the bisection approach for generating utility values.

In the problems we consider, utility values must be assigned not to arbitrary performance levels, but to the given performance levels of existing alternatives. Thus, the domain (or the current sub-interval of the domain) must be split at some predefined performance level rather than the middle of the interval. In general, the algorithm can be described by the following recursive procedure:

procedure UtilityBisection($x_{lo}, x_{up}, u(x_{lo}), u(x_{up})$)
  if a performance level exists in $]x_{lo}, x_{up}[$ **then**
    $x_m \leftarrow$ SplitPoint(x_{lo}, x_{up})
    $u(x_m) \leftarrow$ AssignUtility($x_m, x_{lo}, x_{up}, u(x_{lo}), u(x_{up})$)
    UtilityBisection($x_{lo}, x_m, u(x_{lo}), u(x_m)$))
    UtilityBisection($x_m, x_{up}, u(x_m), u(x_{up})$)
  end if
end procedure

Procedure UtilityBisection takes as arguments the upper and lower bounds of an interval for which a utility function is to be generated (x_{lo} and x_{up}), and their utility values ($u(x_{lo})$ and $u(x_{up})$). It splits the interval at some point, generates a utility value for that point, and then is applied recursively to the upper and lower part of the interval, until utility values are assigned to all performance levels.

Procedure SplitPoint determines a performance level at which the interval is split. Several rules for this choice are possible:

1. The performance level which is closest to $(x_{lo} + x_{up})/2$;
2. The median of performance levels between x_{lo} and x_{up};
3. A randomly chosen performance level between x_{lo} and x_{up};
4. A randomly chosen performance level, where the probability of selecting a performance level is proportional to its average distance to neighboring performance levels.

The idea of the last method is to give priority to performance levels in regions where the density of performance levels is low, thus counteracting the increase in the slope of utility functions in regions where performance levels are densely distributed.

Procedure AssignUtility randomly generates the utility value that is assigned to x_m, i.e., it selects a random y-value for each given x-value, in a way that keeps the function monotonic. The most straightforward approach is to generate a uniformly distributed random value between $u(x_{lo})$ and $u(x_{up})$. However, if x_m is not located exactly at $(x_{lo} + x_{up})/2$, this approach might again introduce a bias in the utility function. To counteract this bias, utility values can be assigned so that convex and concave utility functions occur with equal probability. Let

$$z = u(x_{lo})\frac{x_{up} - x_m}{x_{up} - x_{lo}} + u(x_{up})\frac{x_m - x_{lo}}{x_{up} - x_{lo}} \tag{1}$$

be the utility value of x_m corresponding to a linear utility function, and r a random number uniformly distributed in $(0, 1)$. Then the utility value of x_m is generated as

$$u(x_m) = \begin{cases} 2ru(x_{lo}) + (1 - 2r)z & \text{if } r < 0.5 \\ (2r - 1)z + (2 - 2r)u(x_{up}) & \text{if } r \geq 0.5 \end{cases} \tag{2}$$

Figure 2 shows the 30 utility functions generated by the bisection method for the same performance levels as in Figure 1. For this example, we used the proportional random method (method 4 in the list above) for selecting a splitting point, and applied the correction (2). Now, the random utility functions are quite diverse and no strong bias towards convexity appears to be present: there are convex functions, but also concave functions, as well as functions that are neither convex or concave (the presentation will discuss that it is possible to introduce a concavity or convexity constraint, if this is sought).

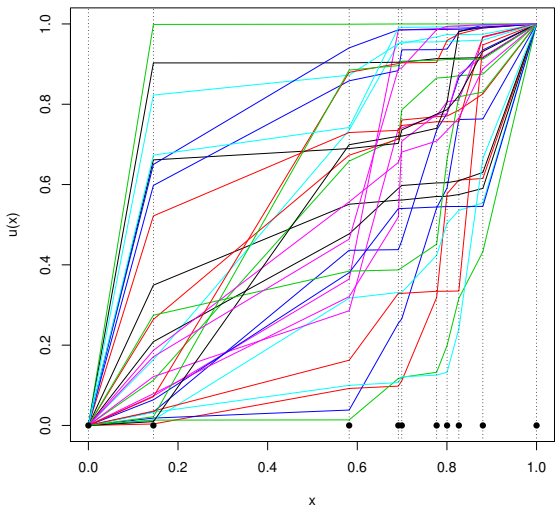

Figure 2. Utility functions generated by the bisection method

3 Evaluation criteria

Our aim is to generate utility functions which are not biased neither to a concave nor to a convex shape, and that are as diverse as possible to cover the entire universe of possible (monotonic) utility functions. We tested the approaches outlined in Section 2 in a computational study, where we compared them using the following criteria.

Consider a utility function defined on the domain (x_{lo}, x_{up}). Any point in the rectangle between the points $(x_{lo}, u(x_{lo}))$, $(x_{lo}, u(x_{up}))$, $(x_{up}, u(x_{up}))$, and $(x_{up}, u(x_{lo}))$ can be a point on a utility function. Denote by $A = (x_{up} - x_{lo})(u(x_{up}) - u(x_{lo}))$ the area of that rectangle. The algorithm generates utility values for $n+1$ attribute values $x_i, i \in \{0, 1, ...n\}$, where $x_0 = x_{lo}$ and $x_n = x_{up}$. We write $u_{\min}(x_i)$ for the lowest utility value assigned to performance level x_i, $u_{\max}(x_i)$ is corresponding highest utility value by, and $\Delta(x_i) = u_{\max}(x_i) - u_{\min}(x_i)$ the difference of these two values. We then define

$$Coverage = \frac{1}{2A} \sum_{i=1}^{n} (x_i - x_{i-1}) \left(\Delta(x_i) + \Delta(x_{i-1})\right) \qquad (3)$$

i.e., *Coverage* measures how much of the possible area containing utility points is actually covered by all the utility functions generated in an experiment. In the two examples shown above, the utility functions generated by the straightforward approach in Figure 1 achieve a coverage of 39.8%, while the bisection approach achieves a coverage of 79.5% and thus a good sampling of the entire area.

We use two measures for avoidance of bias for convex or concave utilities:

1. The absolute difference in the number of points above and below the straight line connecting $u(x_{lo})$ and $u(x_{up})$. The larger this difference, the more the method is biased to either convex or concave functions.
2. The average area below each of the generated utility functions. If utility functions are well balanced between convex and concave shapes, this area on average should be $A/2$.

The absolute difference for the example shown in Figure 1 is 59.2% of all points, while for the example in Figure 2 it is only 3.3%. The average area under the utility function is 32.1% and 51.4% of the total area. Both measures thus clearly show that the utility functions generated by the bisection method exhibit a much better balance between convex and concave functions, as is also clearly visible when comparing Figures 1 and 2.

4 Outlook

We performed extensive simulations of the methods outlined in Section 2. Since our main aim is to avoid a bias in utility functions caused by a skewed distribution of performance levels, performance levels in the simulations were drawn from a distribution with linearly increasing density in the interval $(0, 1)$.

Detailed results of this simulation will be presented at the conference. The main results indicate that the bisection method in general is able to avoid a bias towards a specific shape of generated utility functions even for strongly skewed performance levels. In particular, methods that select split points randomly (with probability proportional to their distance), and that use equation (2) to correct for the split-point not being located at the center of the interval, offer both a good coverage (with coverage levels in the range of 95% and above for problems with 100 performance levels) and good balance between convex and concave utilities. Methods that split at the median, or at points near the mean, perform less well, but are still much better than direct assignment of utility values both in terms of coverage and in avoiding bias.

REFERENCES

[1] Miłosz Kadziński and Tommi Tervonen, 'Robust multi-criteria ranking with additive value models and holistic pair-wise preference statements', *European Journal of Operational Research*, **228**(1), 169 – 180, (2013).

[2] Risto Lahdelma, Joonas Hokkanen, and Pekka Salminen, 'SMAA - stochastic multiobjective acceptability analysis', *European Journal of Operational Research*, **106**(1), 137–143, (1998).

[3] Risto Lahdelma, Kaisa Miettinen, and Pekka Salminen, 'Ordinal criteria in stochastic multicriteria acceptability analysis (SMAA)', *European Journal of Operational Research*, **147**(1), 117 – 127, (2003).

[4] Tommi Tervonen and José Rui Figueira, 'A survey on stochastic multicriteria acceptability analysis methods', *Journal of Multi-Criteria Decision Analysis*, **15**(1-2), 1–14, (2008).

[5] Tommi Tervonen, Gert van Valkenhoef, Nalan Baştürk, and Douwe Postmus, 'Hit-and-run enables efficient weight generation for simulation-based multiple criteria decision analysis', *European Journal of Operational Research*, **224**(3), 552–559, (2013).

[6] Rudolf Vetschera, 'Deriving rankings from incomplete preference information: A comparison of different approaches', *European Journal of Operational Research*, **258**(1), 244–253, (2017).