Decision Making under Interval Uncertainty: What Can and What Cannot Be Computed in Linear Time and in Real Time

O. Kosheleva and V. Kreinovich
University of Texas at El Paso, El Paso, TX 79968, USA, olgak@utep.edu, vladik@utep.edu

Abstract. In engineering, we constantly need to make decisions: which design to select, which parameters to select for this design, etc.

The traditional approach to decision making is based on the assumption that we know all possible consequences of each alternative, and we know the probability of each such consequence. Under this assumption, we can describe a rational decision-making process: to each possible consequence, we assign a numerical values called its utility, and we select the alternative for which the expected value of the utility is the largest.

An important advantage of this approach is that it can be performed in *real time:* if after we made a decision, a new alternative appears, we do not need to repeat the whole analysis again: all we need to do is compare the new alternative with the previously selected ones.

In the past, when we used the same procedures year after year, we accumulated a lot of data about the consequences of different decisions – based from which we could estimate the desired probabilities. Nowadays, with new technologies, new materials constantly emerging, we do not have such detailed information about the consequences of these new technologies. As a result, we often only have partial information about the corresponding probabilities. Different possible probability values result in different values of expected utility. Hence, for each alternative, instead of a single value of expected utility, we have a range (interval) of possible values. We need to make a decision under such interval uncertainty.

In this paper, we describe when we can make decisions under interval uncertainty in linear time and in real time – and when we cannot.

Keywords: decision making; interval uncertainty; real-time computations.

1. Decision Making under Interval Uncertainty: Formulation of the Problem

Decisions are needed in engineering. In engineering, we constantly need to make decisions: which design to select, which parameters to select for this design, etc.

Traditional approach to decision making. The traditional approach to decision making is based on the assumption that we know all possible consequences of each alternative i, and for each alternative i, we know the probability p_{ik} of each such consequence k. Under this assumption, we can describe the usual rational decision-making process (Fishburn, 1988; Luce and Raiffa, 1989; Nguyen et al., 2012; Raiffa, 1997):

- to each possible consequence k of an alternative i, we assign a numerical value called its *utility* u_{ik}, and
- we select the alternative i for which the expected value $u_i \stackrel{\text{def}}{=} \sum_k p_{ik} \cdot u_{ik}$ of the utility is the largest.

If several alternatives have the largest values of the expected utility, then we should generate the list of all these best alternatives. In other words, we need to solve the following problem:

Traditional approach to decision making: precise formulation of the problem.

- we know the values $u_1, \ldots, u_n$ of expected utility corresponding to all possible alternatives;
- we need to generate the list of all alternatives i for which $u_i = \max_j u_j$.

Asymptotically optimal way of solving the above problem: linear-time algorithm. The following natural algorithm can produce the desired list:

- First, we find the largest value $M \stackrel{\text{def}}{=} \max_{1 \le j \le n} u_j$. This is usually done in n steps, by sequentially computing the value $M_i \stackrel{\text{def}}{=} \max_{1 \le j \le i} u_j$ for $i = 1, 2, \ldots, n$: we start with $M_1 = u_1$ and then sequentially compute $M_i = \max(M_{i-1}, u_i)$. The desired value is $M = M_n$.
- Then, we go over all the alternatives $i = 1, \ldots, n$ and select those for which $u_i = M$.

The first stage of this algorithm requires n elementary operations, the second stage also requires n operations, so the total number of elementary operations is $2n$. Algorithms which take $\le C \cdot n$ computational steps are known as *linear-time* algorithms, so the above algorithm is linear time.

It is easy to show that this algorithm is asymptotically optimal in the sense that any algorithm for solving the above problem must take at least linear time. Indeed, to find all best alternatives, we need to handle each of n alternatives at least once: otherwise, if we never process the utility value u_i of some alternative, it may be that this alternative is one of the best – or even it may be the only best alternative. A computational complexity counts elementary arithmetic operations, comparisons, etc. Each elementary operation handles at most two values. Thus, to be able to handle all n values, we need to perform at least $n/2$ elementary operations.

Need for real-time computations. The above algorithm assumes that we know the utility values of all the alternatives. In practice, often, new alternatives are added all the time. When a new alternative is added, we do not want to start the process from scratch, we would like to speed up computations as much as possible by *modifying* the previous list of best alternatives. In other words, we face the following problem:

- we have a list $i_1, \ldots, i_m$ of all alternatives which are the best among the first n;
- we get a new alternative, with utility u_{n+1};

– we want to produce the list of all alternatives which are the best among the first $n+1$ alternatives.

Computational situations in which we need to update the result every time new data appears are known as *real-time* computations.

Asymptotically optimal way of solving the real-time computation problem. The following natural algorithm solves the above real-time computation problem:

– If $u_{n+1} < u_{i_1}$, then we keep the original list of best alternatives.

– If $u_{n+1} = u_{i_1}$, then we add the new alternative $n+1$ to the list of best alternatives.

– Finally, if $u_{n+1} > u_{i_1}$, we replace the original list of best alternatives with a list consisting of a single new alternative $n+1$.

The largest number of computational steps is needed in the last case, when we need to delete all m alternatives from the list. Thus, in the worst-case, this algorithm requires $m+2 = O(m)$ computational steps.

We cannot solve this problem faster, since if $u_{n+1} > u_{i_1}$, we do need to delete all m elements from the original list, and this alone requires $O(m)$ steps. Thus, this algorithm is asymptotically optimal.

Information is often only partial: need to consider interval uncertainty. In practice, we often have only partial information about the probabilities p_{ik} of different consequences. For example, we may know the lower and upper bounds $\underline{p}_{ik}$ and $\overline{p}_{ik}$ for which $\underline{p}_{ik} \le p_{ik} \le \overline{p}_{ik}$. For each alternative i, different possible values p_{ik} from the corresponding intervals $\left[\underline{p}_{ik}, \overline{p}_{ik}\right]$ lead, in general, to different values of the corresponding expected utility $u_i = \sum\limits_k p_{ik} \cdot u_{ik}$. These values form an interval $\mathbf{u}_i = [\underline{u}_i, \overline{u}_i]$ which can be computed by using the standard interval computation techniques (Jaulin et al., 2001; Moore, Kearfott, and Cloud, 2009):

$$[\underline{u}_i, \overline{u}_i] = \sum_k \left[\underline{p}_{ik}, \overline{p}_{ik}\right] \cdot u_{ik},$$

where the product $[\underline{p}, \overline{p}] \cdot u \stackrel{\text{def}}{=} \{p \cdot u : p \in [\underline{p}, \overline{p}]\}$ can be computed as follows:

– if $u \ge 0$, then $[\underline{p}, \overline{p}] \cdot u = [\underline{p} \cdot u, \overline{p} \cdot u]$;

– if $u < 0$, then $[\underline{p}, \overline{p}] \cdot u = [\overline{p} \cdot u, \underline{p} \cdot u]$.

We therefore need to make a decision based on the intervals $[\underline{u}_i, \overline{u}_i]$.

Decision making under interval uncertainty: options. We want to find the best alternatives, i.e., the alternatives for which the expected utility u_i is the largest possible. In the case of interval uncertainty, we do not know the exact values of u_i, we only know the interval of possible values of each u_i. For different values u_i from the corresponding intervals, we may get different lists of best alternatives. So, here, in principle, we have two choices:

- we can produce a list of alternatives which are *necessarily* optimal, i.e., which are optimal no matter which values u_i from the corresponding intervals we choose;
- we can also produce a list of alternatives which are *possibly* optimal, i.e., which are optimal for *some* values u_i from the corresponding intervals,

The problem with the first option is that often, no such alternatives exist: e.g., if $\mathbf{u}_1 = \mathbf{u}_2 = [1, 2]$, then for $u_1 = 2$ and $u_1 = 1$, the first alternative is the only best one, while for $u_1 = 1$ and $u_2 = 2$, the second alternative is the only best one. It is therefore desirable to produce *both* lists.

Another idea is to use *Hurwicz optimism-pessimism criterion* (see, e.g., (Luce and Raiffa, 1989)), and produce alternatives for which, for some $\alpha \in [0, 1]$, the value $u_i = \alpha \cdot \overline{u}_i + (1 - \alpha) \cdot \underline{u}_i$ is the largest possible. When $\alpha = 1$, this means that we only consider the most optimistic outcomes $\overline{u}_i$; when $\alpha = 0$, this means that we only consider the most pessimistic outcomes $\underline{u}_i$; in general, we combine the optimistic and the pessimistic outcomes – that is why this is called optimism-pessimism criterion.

If we fix α, then, from the computational viewpoint, the problem is similar to what we have without interval uncertainty: producing the list of alternatives for which the corresponding value u_i is the largest possible. A non-trivial problem emerges if we do not know the value α beforehand, and we want to be able to make recommendations corresponding to all possible values α.

What we do in this paper. In this paper, we analyze the computational complexity of decision making under interval uncertainty: we analyze which problems can be solved in linear time and in real time, and which cannot.

2. Decision making under Interval Uncertainty: Linear-Time Algorithms

Necessarily optimal alternatives: reminder. An alternative i is necessarily optimal if for every $j \neq i$, we have $u_i \geq u_j$ for all possible values $u_i \in [\underline{u}_i, \overline{u}_i]$ and $u_j \in [\underline{u}_j, \overline{u}_j]$. In particular, this means that we should have $\underline{u}_i \geq \overline{u}_j$ for all $j \neq i$. Once this inequality is satisfied, one can easily check that every value $u_i \geq \underline{u}_i$ is larger than or equal to any value $u_j \leq \overline{u}_j$.

Thus, an alternative i is necessarily optimal if and only if $\underline{u}_i \geq \overline{u}_j$ for all $j \neq i$.

Possibly optimal alternatives: reminder. An alternative i is possibly optimal if for every $j \neq i$, we have $u_i \geq u_j$ for some values $u_i \in [\underline{u}_i, \overline{u}_i]$ and $u_j \in [\underline{u}_j, \overline{u}_j]$. From $\underline{u}_j \leq u_j \leq u_i \leq \overline{u}_i$, we conclude that $\underline{u}_i \leq \overline{u}_j$. Vice versa, if $\underline{u}_i \leq \overline{u}_j$, then some possible value u_i (namely, $\underline{u}_i$) is smaller than or equal to some possible value u_j (namely, the value $\overline{u}_j$).

In particular, this means that we should have $\underline{u}_i \geq \overline{u}_j$ for all $j \neq i$. Once this inequality is satisfied, one can easily check that every value $u_i \geq \underline{u}_i$ is larger than or equal to any value $u_j \leq \overline{u}_j$.

Thus, an alternative i is possibly optimal if and only if $\overline{u}_i \geq \underline{u}_j$ for all $j \neq i$.

Comment. We may have several possibly optimal alternatives, but the only possibility to have several necessarily optimal alternatives i, j, ..., i.e., to have $\underline{u}_i \geq \overline{u}_j$ and $\underline{u}_j \geq \overline{u}_i$ is to have $\underline{u}_i \geq \overline{u}_j \geq \underline{u}_j \geq \overline{u}_i \geq \underline{u}_i$, i.e., to have $[\underline{u}_i, \overline{u}_i] = [\underline{u}_j, \overline{u}_j] = \{\underline{u}_i\}$. In other words, for each such alternative, the expected utility value is known exactly, with no uncertainty.

Resulting straightforward algorithm and its limitations. Based on the above conclusions, we can get an algorithm for generating lists of necessarily optimal and possibly optimal alternatives: for each alternative i, to check whether this alternative belongs to the corresponding list, we compare it with all other alternatives $j \neq i$. If the corresponding inequality holds for all these $j \neq i$, then the alternative i is included in the desired list, otherwise the alternative i is not included in this list.

The problem with this approach is that to form a list, we compare each of n alternatives with each of $n-1$ remaining ones. Thus, this algorithm performs $n \cdot (n-1) = O(n^2)$ comparisons – and so, requires quadratic time, which for large n is much longer than linear time.

Can we produce these lists faster?

Towards a linear time algorithm for producing the list of possibly best alternatives. For decision making without uncertainty, since $u_i \geq u_i$, the requirement that $u_i \geq u_j$ for all $j \neq i$ is equivalent to requiring that $u_i \geq u_j$ for all j and is, thus, equivalent to $u_i = \max\limits_j u_j$. In other words, it is sufficient to find the largest possible value M of all the values u_j and then, instead of comparing all alternatives with each other (which would take quadratic time), to compare each of them with this largest value M.

Similarly, since $\overline{u}_i \geq \underline{u}_i$, the requirement that $\overline{u}_i \geq \underline{u}_j$ for all $j \neq i$ is equivalent to requiring that $\overline{u}_i \geq \underline{u}_j$ for all j and is, thus, equivalent to $\overline{u}_i = \max\limits_j \underline{u}_j$. In other words, it is sufficient to find the largest possible value $\underline{M}$ of all the values $\underline{u}_j$ and then, instead of comparing all alternatives with each other (which would take quadratic time), to compare each of them with this largest value $\underline{M}$.

Thus, we arrive at the following algorithm.

A linear-time algorithm for producing the list of possibly optimal alternatives.

- First, we find the largest value $\underline{M} \stackrel{\text{def}}{=} \max\limits_{1 \leq j \leq n} \underline{u}_j$. This can be done by sequentially computing the values $\underline{M}_i \stackrel{\text{def}}{=} \max\limits_{1 \leq j \leq i} \underline{u}_j$ for $i = 1, 2, \ldots, n$: we start with $\underline{M}_1 = \underline{u}_1$ and then sequentially compute $\underline{M}_i = \max(\underline{M}_{i-1}, \underline{u}_i)$. The desired value is $\underline{M} = \underline{M}_n$.
- Then, we go over all the alternatives $i = 1, \ldots, n$ and select those for which $\overline{u}_i \geq \underline{M}$.

Both stages take linear time, so the above algorithm is also linear-time.

Towards a linear-time algorithm for producing the list of necessarily best alternatives. In this case, it is not enough to know the largest possible value $\overline{M} = \max\limits_j \overline{u}_j$, but we will show that it is sufficient to know this value and the second largest $\overline{S} \leq \overline{M}$ of these values. Indeed, if we know both the largest and the second largest values, then, for each i, we can detect whether the alternative i is necessarily optimal as follows:

- Let us first consider the case when $\overline{u}_i < \overline{M}$. By definition, $\overline{M} = \overline{u}_j$ for some alternative j – which is thus different from i, so $\overline{u}_i < \overline{u}_j$. Therefore, for $\underline{u}_i \leq \overline{u}_i$, we also have $\underline{u}_i < \overline{u}_j$ and hence, this alternative i is *not* necessarily optimal.
- The only remaining case is when $\overline{u}_i = \overline{M}$. In this case, we have to consider two subcases: when $\overline{S} < \overline{M}$, and when $\overline{S} = \overline{M}$.

- In the first subcase, the inequality $\overline{S} < \overline{M}$ means that the alternative i is the only alternative with $\overline{u}_i = \overline{M}$, and thus, $\max\limits_{j \neq i} \overline{u}_j = \overline{S}$. So, checking whether $\underline{u}_j \geq \max\limits_{j \neq i} \overline{u}_j$ is equivalent to checking whether $\underline{u}_i \geq \overline{S}$.
- In the second subcase, the inequality $\overline{S} = \overline{M}$ means that there are other alternatives $j \neq i$ for which $\overline{u}_j = \overline{M}$. In this case, $\max\limits_{j \neq i} \overline{u}_j = \overline{M}$. So, checking whether $\underline{u}_j \geq \max\limits_{j \neq i} \overline{u}_j$ is equivalent to checking whether $\underline{u}_i \geq \overline{M}$. Since in this subcase, $\overline{M} = \overline{S}$, this is also equivalent to checking whether $\underline{u}_i \geq \overline{S}$.

Thus, in both subcases, we check whether $\underline{u}_i \geq \overline{S}$.

So, we arrive at the following linear-time algorithm.

A linear-time algorithm for producing the list of necessarily optimal alternatives.

– First, we find the largest value $\overline{M} \stackrel{\text{def}}{=} \max\limits_{1 \leq j \leq n} \overline{u}_j$ and the second largest value $\overline{S}$. This can be done by sequentially computing the largest $\overline{M}_i \stackrel{\text{def}}{=} \max\limits_{1 \leq j \leq i} \overline{u}_j$ and the second largest $\overline{S}_i$ among the values $\overline{u}_1, \ldots, \overline{u}_i$, for $i = 2, 3, \ldots, n$. First, we compute $\overline{M}_2 = \max(\overline{u}_1, \overline{u}_2)$ and $\overline{S}_2 = \min(\overline{u}_1, \overline{u}_2)$. Then, for $i = 3, 4, \ldots, n$, we update these values as follows:

- if $\overline{u}_i \geq \overline{M}_{i-1}$, then we take $\overline{M}_i = \overline{u}_i$ and $\overline{S}_i = \overline{M}_{i-1}$;
- if $\overline{S}_{i-1} < \overline{u}_i < \overline{M}_{i-1}$, then we take $\overline{M}_i = \overline{M}_{i-1}$ and $\overline{S}_i = \overline{u}_i$;
- finally, if $\overline{u}_i \leq \overline{S}_{i-1}$, then we keep both values unchanged: $\overline{M}_i = \overline{M}_{i-1}$ and $\overline{S}_i = \overline{S}_{i-1}$.

Finally, we take $\overline{M} = \overline{M}_n$ and $\overline{S} = \overline{S}_n$.

– Then, we go over all the alternatives $i = 1, \ldots, n$ and select those for which $\overline{u}_i = \overline{M}$ and $\underline{u}_i \geq \overline{S}$.

The first stage of this algorithm requires $O(n)$ elementary operations, the second stage also requires $O(n)$ operations, so the above algorithm is indeed linear-time.

3. Decision making under Interval Uncertainty: Real-Time Algorithms

General idea. The possibility for real-time algorithms comes from the easy-to-see observation that if, after adding a new alternative, one of the old alternatives remains possibly or necessarily optimal, then this alternative was possibly (correspondingly, necessarily) optimal before as well. Thus, to update the desired list, it is sufficient to analyze the previous list – and the new alternative.

Case of possibly optimal alternatives: asymptotically optimal way of solving the real-time computation problem. Suppose that:

– we have a list $i_1, \ldots, i_m$ of all alternatives which are possibly optimal among the first n;

- we know the auxiliary value $\underline{M}_n = \max(\underline{u}_1, \ldots, \underline{u}_n)$ which was used to find the possibly optimal alternatives;
- we get a new alternative, with utility interval $[\underline{u}_{n+1}, \overline{u}_{n+1}]$;
- we want to produce the list of all alternatives which are possibly optimal among the first $n+1$ alternatives – and to update the auxiliary value $\underline{M}$.

The following natural algorithm solves the above real-time computation problem:

- If $\underline{u}_{n+1} \le \underline{M}_n$, then the auxiliary value $\underline{M}$ stays the same: $\underline{M}_{n+1} = \underline{M}_n$. Here:
  - If $\overline{u}_{n+1} < \underline{M}_n$, then we keep the original list of possibly optimal alternatives.
  - If $\overline{u}_{n+1} \ge \underline{M}_n$, then we add the new alternative $n+1$ to the list of possibly optimal alternatives.
- If $\underline{u}_{n+1} > \underline{M}_n$, then we update the auxiliary value: $\underline{M}_{n+1} = \underline{u}_{n+1}$. Out of the original list $\{i_1, \ldots, i_m\}$ of possibly optimal alternatives, we only keep those for which $\overline{u}_{i_k} \ge \underline{M}_{n+1}$. To thus reduced list, we add a new alternative $n+1$.

This algorithm requires $O(m)$ steps and is, therefore, asymptotically optimal – since even in the case of no uncertainty, we may need $O(m)$ steps to delete the original list.

Case of necessarily optimal alternatives: asymptotically optimal way of solving the real-time computation problem. Suppose that:

- we have a list $i_1, \ldots, i_m$ of all alternatives which are necessarily optimal among the first n;
- we know two auxiliary values: $\overline{M}_n$ is the largest of the values $\overline{u}_1, \ldots, \overline{u}_n$, and $\overline{S}_n$ is the second largest of these values;
- we get a new alternative, with utility interval $[\underline{u}_{n+1}, \overline{u}_{n+1}]$;
- we want to produce the list of all alternatives which are possibly optimal among the first $n+1$ alternatives – and to update the auxiliary values $\underline{M}_n$ and $\underline{S}_n$.

The updating part is straightforward:

- if $\overline{u}_{n+1} \ge \overline{M}_n$, then we take $\overline{M}_{n+1} = \overline{u}_{n+1}$ and $\overline{S}_{n+1} = \overline{M}_n$;
- if $\overline{S}_n < \overline{u}_{n+1} < \overline{M}_n$, then we take $\overline{M}_{n+1} = \overline{M}_n$ and $\overline{S}_{n+1} = \overline{u}_{n+1}$;
- finally, if $\overline{u}_{n+1} \le \overline{S}_n$, then we keep both auxiliary values unchanged: $\overline{M}_{n+1} = \overline{M}_n$ and $\overline{S}_{n+1} = \overline{S}_n$.

Then:

- out of the original list $\{i_1, \ldots, i_m\}$, we only keep those alternatives i_k for which $\overline{u}_{i_k} = \overline{M}_{n+1}$ and $\underline{u}_{i_k} \ge \overline{S}_{n+1}$, and

– we add the new alternative $n+1$ to the desired list if $\overline{u}_{n+1} = \overline{M}_{n+1}$ and $\underline{u}_{n+1} \geq \overline{S}_{n+1}$.

This algorithm requires $O(m)$ steps and is, thus, asymptotically optimal.

4. Case of Hurwicz Optimism-Pessimism Criterion with Unknown α: No Linear Time Algorithm Is Possible

Problem: reminder.

– We have n alternatives, about which we only know the intervals $[\underline{u}_i, \overline{u}_i]$ of possible values of expected utility.

– We want to be able to find, for each possible values $\alpha \in [0,1]$, which alternative(s) i is the best for each α, i.e., for which $u_i \stackrel{\text{def}}{=} \alpha \cdot \overline{u}_i + (1-\alpha) \cdot \underline{u}_i \geq u_j = \alpha \cdot \overline{u}_j + (1-\alpha) \cdot \underline{u}_j$ for all $j \neq i$.

Analysis of the problem. The condition that for a given α, the alternative i is the best means that $u_i \geq u_j$ for all $j \neq i$, i.e., that $\alpha \cdot \overline{u}_i + (1-\alpha) \cdot \underline{u}_i \geq \alpha \cdot \overline{u}_j + (1-\alpha) \cdot \underline{u}_j$. Each such inequality is a linear inequality in terms of α, which can be represented as $\alpha \cdot a \geq b$ for some a and b. Depending on the sign of a, this is equivalent to either $\alpha \geq c$ or to $\alpha \leq c$ for some constant c. Thus, for each i and j, the set of all the values α which satisfy this inequality is an interval $[c,1]$ or $[0,c]$. The set of values α for which the alternative i is optimal is the intersection of these intervals - and thus, also an interval.

Since for every α, some alternative is optimal, intervals corresponding to optimality of different alternatives fill the whole interval $[0,1]$ of possible values of α. Thus, their endpoints

$$\alpha_0 \stackrel{\text{def}}{=} 0 < \alpha_1 < \ldots < \alpha_k < \ldots < \alpha_N = 1$$

divide the interval $[0,1]$ into subintervals $[\alpha_k, \alpha_{k+1}]$ on each of which certain alternative(s) is optimal. (The endpoints of these intervals is where two expressions u_i and u_j are equal, i.e., where $\alpha \cdot \overline{u}_i + (1-\alpha) \cdot \underline{u}_i = \alpha \cdot \overline{u}_j + (1-\alpha) \cdot \underline{u}_j$.)

We therefore arrive at the following formulation of the problem.

Precise formulation of the problem.

– We know n intervals $[\underline{u}_1, \overline{u}_1]$, …, $[\underline{u}_n, \overline{u}_n]$.

– We want to compute the values $\alpha_0 = 0 < \alpha_1 < \ldots < \alpha_k < \ldots < \alpha_N = 1$ and the lists $L_0, \ldots, L_{N-1} \subseteq \{1, \ldots, n\}$ such that for all $\alpha \in [\alpha_k, \alpha_{k+1}]$, alternatives from the list L_k are optimal for this α.

In general, this problem cannot be solved in linear time: a proof. Let us prove that this problem cannot be solved in linear time. Let us take n values $x_1, \ldots, x_n \in [0,1]$, and let us form n intervals $\mathbf{u}_i = [1-x_i^2, 2-(1-x_i)^2]$. For each α, the function $u(x) = \alpha \cdot (2-(1-x)^2) + (1-\alpha) \cdot (1-x^2)$ attains its maximum when $u'(x) = 0$, i.e., when $2\alpha \cdot (1-x) - 2(1-\alpha) \cdot x = 0$ and $x = \alpha$. Thus, for

$\alpha = x_i$, the value u_i is larger than all the values u_j corresponding to all other alternatives $j \neq i$. So, in the interval containing $\alpha = x_i$, the corresponding list L consists of a single alternative i.

Thus, the sequence of lists L_0, L_1, ..., contains the alternatives sorted in the increasing order of x_i. Hence, if we could solve the above problem in linear time, we would be able to sort any n real numbers in linear time. It is known, however, that sorting of n numbers requires at least $n \cdot \ln(n)$ computational steps (Cormen et al., 2009). This proves that the above problem cannot be solved in linear time.

Acknowledgements

This work was supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721.

References

Cormen, T. H., et al. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 2009.

Fishburn, P. C. *Nonlinear Preference and Utility Theory*. John Hopkins Press, Baltimore, Maryland, 1988.

Jaulin, L., et al. *Applied Interval Analysis*. Springer Verlag, London, 2001.

Luce, R. D. and R. Raiffa. *Games and Decisions: Introduction and Critical Survey*. Dover, New York, 1989.

Moore, R. E., R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. SIAM Press, Philadelphia, Pennsylvania, 2009.

Nguyen, H. T., et al. *Computing Statistics under Interval and Fuzzy Uncertainty*, Springer Verlag, Berlin, Heidelberg, 2012.

Raiffa, H. *Decision Analysis*, McGraw-Hill, Columbus, Ohio, 1997.