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

Olga Kosheleva and Vladik Kreinovich
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
olgak@utep.edu, vladik@utep.edu

http://www.cs.utep.edu/vladik/olgavita.html
http://www.cs.utep.edu/vladik

1. Traditional Approach to Decision Making

- In engineering, we make decisions: which design to select, which parameters to select for this design.
- 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:
  - to each possible consequence k of an alternative i, we assign a numerical value called its *utility* u_{ik};
  - we select the alternative(s) i for which the expected value $u_i \stackrel{\text{def}}{=} \sum_k p_{ik} \cdot u_{ik}$ of the utility is the largest.

2. Traditional Decision Making (cont-d)

- We know the values $u_1, \ldots, u_n$ corr. to all alternatives.
- We need to generate the list of all alternatives i for which $u_i = \max\limits_j u_j$. Natural idea:
  - Start with $M_1 = u_1$ and sequentially compute $M_i = \max(M_{i-1}, u_i)$, until we reach $M_n = M \stackrel{\text{def}}{=} \max\limits_j u_j$.
  - Then, we go over all the alternatives $i = 1, \ldots, n$ and select those for which $u_i = M$.
- This algorithm is linear-time.
- This algorithm is asymptotically optimal:
  - we must handle all n alternatives – else we may miss the best one;
  - each elementary operation processes ≤ 2 numbers;
  - thus, we need at least $n/2$ operations.

3. 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. In this case:
  - we do not want to start the process from scratch;
  - we would like to speed up computations by *modifying* the previous list of best alternatives.
- We face the following problem of *real-time* computations:
  - 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.

4. Asymptotically Optimal Way of Solving the 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 single new alternative $n+1$.
- The largest number of computational steps is needed 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.
- Thus, this algorithm is asymptotically optimal.

5. 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 $\underline{p}_{ik} \le p_{ik} \le \overline{p}_{ik}$.
- Different $p_{ik} \in \left[\underline{p}_{ik}, \overline{p}_{ik}\right]$ lead, in general, to different values of the expected utility $u_i = \sum_k p_{ik} \cdot u_{ik}$.
- These values form an interval $\mathbf{u}_i = [\underline{u}_i, \overline{u}_i]$ which can be computed by using the standard interval computation

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

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

6. Decision Making Under Interval Uncertainty: Options

- We want to find the best alternatives, i.e., the alternatives i for which $u_i \to \max$.
- In the case of interval uncertainty, for different values u_i, 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 (for all u_i);
  - we can also produce a list of alternatives which are *possibly* optimal (for *some* u_i),
- If $\mathbf{u}_1 = \mathbf{u}_2 = [1, 2]$, no i is necessarily optimal:
  - for $u_1 = 2$ and $u_1 = 1$, the 1st is the best,
  - for $u_1 = 1$ and $u_2 = 2$, the 2nd is the best.
- So, it is desirable to produce *both* lists.

7. Options (cont-d)

- Another idea is to use *Hurwicz optimism-pessimism criterion*, 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.
- If we fix α, then, from the computational viewpoint, we have the same problem as without uncertainty.
- A problem is non-trivial if we want to be able to make recommendations corresponding to all possible α.
- In this talk, we analyze analyze which problems can be solved in linear time and in real time.

8. Analysis of the Problem

- An alternative i is necessarily optimal if for every $j \neq i$, we have $u_i \geq u_j$ for all $u_i \in [\underline{u}_i, \overline{u}_i]$ and $u_j \in [\underline{u}_j, \overline{u}_j]$.
- In particular, this implies $\underline{u}_i \geq \overline{u}_j$ for all $j \neq i$.
- Vice versa, if $\underline{u}_i \geq \overline{u}_j$, then $u_i \geq \underline{u}_i \geq \overline{u}_j \geq u_j$ and $u_i \geq u_j$.
- Thus, an alternative i is necessarily optimal if and only if $\underline{u}_i \geq \overline{u}_j$ for all $j \neq i$.
- If i and j are both necessarily optimal, then $\underline{u}_i \geq \overline{u}_j \geq \underline{u}_j \geq \overline{u}_i$, so $\underline{u}_i = \overline{u}_i = \underline{u}_j = \overline{u}_j$.
- In other words, for each such alternative, the expected utility value is known exactly, with no uncertainty.

9. Analysis (cont-d) and the Resulting Straightforward Algorithm

- An alternative i is possibly optimal if for every $j \neq i$, we have $u_i \geq u_j$ for some $u_i \in [\underline{u}_i, \overline{u}_i]$ and $u_j \in [\underline{u}_j, \overline{u}_j]$.
- If $u_j \leq u_i$, then 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 $u_i = \underline{u}_i$ is smaller than or equal to $u_j = \overline{u}_j$.
- Thus, an alternative i is possibly optimal if and only if $\overline{u}_i \geq \underline{u}_j$ for all $j \neq i$.
- Resulting algorithm:
  - for each i, we compare it with all $j \neq i$;
  - if the corr. inequality ($\underline{u}_i \geq \overline{u}_j$ or $\overline{u}_i \geq \underline{u}_j$) holds for all $j \neq i$, then i is included in the corr. list.
- This takes $O(n^2)$ comparisons; can we do it faster?

10. Towards a Linear Time Algorithm for Producing the List of Possibly Best Alternatives

- *Reminder:* i is possibly optimal if $\overline{u}_i \geq \underline{u}_j$ for all $j \neq i$.
- *Observation:* $\overline{u}_i \geq \underline{u}_j$.
- *Conclusion:* i is possible optimal if $\overline{u}_i \geq \underline{u}_j$ for *all* j, i.e., if $\overline{u}_i \geq \underline{M}_n \stackrel{\text{def}}{=} \max\limits_{1 \leq j \leq n} \underline{u}_j$.
- *Resulting algorithm:*
  - First, we compute $\underline{M}_1 = \underline{u}_1$ and $\underline{M}_i = \max(\underline{M}_{i-1}, \underline{u}_i)$ for $i = 2, \ldots, n$.
  - 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*.

11. Necessarily Optimal: Analysis

- *Reminder:* i is necessarily optimal if $\underline{u}_i \geq \overline{u}_j$.
- Let $\overline{M} = \max_j \overline{u}_j$ and let $\overline{S}$ be the second largest of $\overline{u}_j$.
- If $\overline{u}_i < \overline{M}$, then, since $\overline{M} = \overline{u}_j$ for some $j \neq i$, we have $\overline{u}_i < \overline{u}_j$, so i is *not* necessarily optimal; so, $\overline{u}_i = \overline{M}$.
- If $\overline{S} < \overline{M}$, then i is the only s.t. $\overline{u}_i = \overline{M}$, so $\max_{j \neq i} \overline{u}_j = \overline{S}$.
- So, checking whether $\underline{u}_j \geq \max_{j \neq i} \overline{u}_j$ is equivalent to checking whether $\underline{u}_i \geq \overline{S}$.
- If $\overline{S} = \overline{M}$, then there exists $j \neq i$ such that $\overline{u}_j = \overline{M}$, hence $\max_{j \neq i} \overline{u}_j = \overline{M}$.
- So, checking $\underline{u}_i \geq \max_{j \neq i} \overline{u}_j$ is equivalent to checking whether $\underline{u}_i \geq \overline{M} = \overline{S}$.
- Thus, in both cases, we check whether $\underline{u}_i \geq \overline{S}$.

12. A Linear-Time Algorithm for Producing the List of Necessarily Optimal Alternatives

- First, we find the largest value $\overline{M} \stackrel{\text{def}}{=} \max\limits_{1 \le j \le n} \overline{u}_j$ and the second largest value $\overline{S}$.
- We do it by sequentially computing the largest $\overline{M}_i$ and the second largest $\overline{S}_i$ among $\overline{u}_1, \ldots, \overline{u}_i$.
- 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 $\overline{u}_i \ge \overline{M}_{i-1}$, 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 $\overline{u}_i \le \overline{S}_{i-1}$, 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 $\overline{S} = \overline{S}_n$.

13. Linear-Time Algorithm (cont-d)

- *Reminder:* first, we find the largest value $\overline{M} \stackrel{\text{def}}{=} \max\limits_{1 \le j \le n} \overline{u}_j$ and the second largest value $\overline{S}$.
- Then, we go over all the alternatives $i = 1, \ldots, n$ and select those for which $\overline{u}_i = \overline{M}$ and $\underline{u}_i \ge \overline{S}$.
- The first stage of this algorithm requires $O(n)$ elementary operations.
- The second stage also requires $O(n)$ operations.
- *Conclusion:* the above algorithm is indeed linear-time.

14. Decision making under Interval Uncertainty: Main Idea Behind Real-Time Algorithms

- *Observation:*
  - if, after adding a new alternative, one of the old alternatives remains possibly or necessarily optimal,
  - then this old alternative was possibly (correspondingly, necessarily) optimal before as well.
- Thus, to update the desired list, it is sufficient:
  - to analyze all elements of the previous list, and
  - to analyze the new alternative.

15. Possibly Optimal Alternatives: Formulation of the Real-Time Computation Problem

- *We know:*
  - a list $i_1, \ldots, i_m$ of all alternatives which are possibly optimal among the first n;
  - the auxiliary value $\underline{M}_n = \max(\underline{u}_1, \ldots, \underline{u}_n)$;
  - the utility interval $[\underline{u}_{n+1}, \overline{u}_{n+1}]$ describing the new alternative.
- *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 $\underline{M}$.

16. Possibly Optimal Alternatives: Solving the Real-Time Computation Problem

- If $\underline{u}_{n+1} \le \underline{M}_n$, then $\underline{M}_{n+1} = \underline{M}_n$. 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 $\{i_1, \ldots, i_m\}$, 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.
- We may need $O(m)$ steps to delete the original list.
- So, this algorithm is asymptotically optimal.

17. Necessarily Optimal Alternatives: Formulation of the Real-Time Computation Problem

- *We know:*
  - the list $i_1, \ldots, i_m$ of all alternatives which are necessarily optimal among the first n;
  - 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;
  - the utility interval $[\underline{u}_{n+1}, \overline{u}_{n+1}]$ describing the new alternative.
- *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 $\underline{S}_n$.

18. Necessarily Optimal Alternatives: Solving the Real-Time Computation Problem

- The updating part is straightforward:
  - if $\overline{u}_{n+1} \geq \overline{M}_n$, 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 $\overline{u}_{n+1} \leq \overline{S}_n$, 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 only i_k for which $\overline{u}_{i_k} = \overline{M}_{n+1}$ and $\underline{u}_{i_k} \geq \overline{S}_{n+1}$, 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 takes $O(m)$ steps and is, thus, optimal.

19. Hurwicz Optimism-Pessimism Criterion with Unknown α: Formulation of the Problem

- *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 find:* for each possible values $\alpha \in [0, 1]$, which alternative(s) i is the best for this α:

  $u_i \stackrel{\text{def}}{=} \alpha \cdot \overline{u}_i + (1-\alpha) \cdot \underline{u}_i \geq u_j = \alpha \cdot \overline{u}_j + (1-\alpha) \cdot \underline{u}_j$ for all $j \neq i$.

- Each ineq. is linear in α, its solutions form an interval.
- For each i, the set of values i for which u_i is the best is an intersection of intervals – an interval $[\alpha_k, \alpha_{k+1}]$.
- *We want to find:* values $\alpha_0 = 0 < \alpha_1 < \ldots < \alpha_k < \ldots < \alpha_N = 1$ and the lists L_k optimal for $\alpha \in [\alpha_k, \alpha_{k+1}]$.

20. In General, This Problem Cannot Be Solved in Linear Time: A Proof

- Let us take n values $x_1, \ldots, x_n \in [0, 1]$, and let us form n intervals $\mathbf{u}_i = [1 - x_i^2, 2 - (1 - x_i)^2]$.
- The function $u(x) = \alpha \cdot (2 - (1 - x)^2) + (1 - \alpha) \cdot (1 - x^2)$ attains its max when $u'(x) = 0$, i.e., when

$$2\alpha \cdot (1 - x) - 2(1 - \alpha) \cdot x = 0 \text{ and } x = \alpha.$$

- 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, $\ldots$, contains the alternatives sorted in the increasing order of x_i.
- Hence, if we could solve our problem in linear time, we would be able to to sort any n numbers in linear time.

21. Acknowledgements

This work was supported in part by the National Science Foundation grants:

- HRD-0734825 and HRD-1242125 (Cyber-ShARE Center of Excellence) and
- DUE-0926721.