Why Copulas Have Been Successful in Many Practical Applications: A Theoretical Explanation Based on Computational Efficiency

Vladik Kreinovich[1], Hung T. Nguyen[2,3],
Songsak Sriboonchitta[3], and Olga Kosheleva[4]

[1]Department of Computer Science, [4]Department of Teacher Education
University of Texas at El Paso, El Paso, Texas 79968, USA
vladik@utep.edu, olgak@utep.edu,
[2]Department of Mathematical Sciences, New Mexico State University
Las Cruces, New Mexico 88003, USA, hunguyen@nmsu.edu
[3]Faculty of Economics, Chiang Mai University
Chiang Mai, Thailand, songsakecon@gmail.com

1. Introduction

- In many practical problems, we deal with joint distributions of several quantities (multi-D distributions).
- There are many different ways to represent such a distribution in a computer.
- In many practical applications, it turns out to be beneficial to use a representation in which we store:
  - the marginal distributions (that describe the distribution of each quantity) and
  - a copula (that describe the relation between different quantities; definitions are given below).
- While this representation is, in many cases, empirically successful, this empirical success is largely a mystery.
- We explain this success by showing that the copula representation is the most computationally efficient.

2. How to Represent Probability Distributions: Idea

- The main purpose of knowledge is to make decisions.
- According to decision theory, a consistent decision comes from the following:
  - we assign a numerical value u (called *utility*) to each possible consequence, and then
  - we select a decision for which the expected value $E[u]$ of utility is the largest possible.
- We should select a representation that facilitates decision making.
- So, we need a representation that allows us to compute the expected utility as efficiently as possible.

3. Case of 1-D Probability Distributions

- Example – transportation: we want to get from point A to point B as fact as possible.
- Often, a small increase in travel time x leads to a small decrease in utility $u(x)$: $u(x)$ is smooth.
- Usually, we can predict x with some accuracy, so the actual x is in a small vicinity of the predicted value x_0.
- In this vicinity:

$$u(x) = u(x_0) + u'(x_0)\cdot(x - x_0) + \frac{1}{2}\cdot u''(x_0)\cdot(x - x_0)^2 + \ldots, \text{ so}$$

$$E[u] = u(x_0) + u'(x_0)\cdot E[x - x_0] + \frac{1}{2}\cdot u''(x_0)\cdot E[(x - x_0)^2] + \ldots$$

- So, to compute $E[u]$, it is sufficient to know the first few moments of the probability distribution.

4. 1-D Distributions (cont-d)

- Sometimes, e.g., if we are driving to the airport to take a flight – a small delay can make us miss a flight.
- In such situations, we have a threshold x_t such that:
  - the utility is high for $x \le x_t$ and
  - low for $x > x_t$.
- The difference between two high values is much smaller than between high and low values.
- Thus, we can simply say that $u = u^+$ for $x \le x_t$ and $u = u^- < u^+$ for $x > x_t$.
- In this case, $E[u] = u^- + (u^+ - u^-) \cdot F(x_t)$, where $F(x_t) = \text{Prob}(x \le x_t)$ is the probability that $x \le x_t$.
- So, to deal with situations of this type, we need to know the cdf $F(x)$.

5. 1-D Distributions: Conclusion

- Our analysis shows that in the 1-D case, to compute the expected utilities, we need to know:
  - the cdf *and*
  - the moments.
- Moments can be computed based on cdf, as

$$E[(x-x_0)^k] = \int (x-x_0)^k \, dF(x).$$

- So, it is thus sufficient to have a cdf; hence:
  - the most appropriate way to represent a 1-D probability distribution in the computer is
  - to store the values of its cumulative distribution function $F(x)$.

6. Multi-D Case

- When small changes in x_i lead to small changes in $u(x_1, \ldots, x_n)$, it's enough to know first few moments.
- In the situations of the second type, we want all the values not to exceed appropriate thresholds; e.g.:
  - the travel time does not exceed T_0, and
  - the overall cost of all the tolls does not exceed C_0.
- So, it is desirable to know the following probabilities – that form the corresponding multi-D cdf:

$$F(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \text{Prob}(X_1 \le x_1 \,\&\, \ldots \,\&\, X_n \le x_n).$$

- Hence, in the multi-D case, we need to compute both the moments and the multi-D cdf.
- Since the moments can be computed based on the cdf, it is thus sufficient to represent a cdf.

7. From 1-D to Multi-D

- The above analysis of is appropriate when we acquire all our knowledge about the probabilities in one step.
- However, often, first we are interested in each x_i, so we first learn the marginals $F_i(x_i)$.
- Later, we become interested in the relation between these x_i, so we need the cdf $F(x_1, \ldots, x_n)$.
- But storing $F(x_1, \ldots, x_n)$ and marginals wastes memory: marginals can be reconstructed from F.
- An alternative is to store a *copula*, i.e., a function $C(x_1, \ldots, x_n)$ s.t.

$$F(x_1, \ldots, x_n) = C(F_1(x_1), \ldots, F_n(x_n)).$$

- Copula representation avoids memory waste, but is it optimal? is it the only optimal one?
- Let us formulate this question in precise terms.

8. What is an Algorithm: Reminder

- We want to be able, given the marginals and the additional function(s), to reconstruct $F(x_1, \ldots, x_n)$.
- This reconstruction has to be done by a computer *algorithm*, a sequence of steps, in each of which we:
  - either apply some operation ($+$, $-$, sin, given function) to previously computed values,
  - or decide where to go further (or stop).
- In computations, we can use inputs, previous computation results, and constants (including $\pm\infty$).
- We can also use NaN (undefined): if one of the inputs is undefined, the result is also undefined.

9. What Is An Algorithm: Definition

- Let F be a finite list of functions $f_i(z_1, \ldots, z_{n_i})$.
- Let $v_1, \ldots, v_m$ be a finite list of real-valued variables called *inputs*.
- Let $a_1, \ldots, a_p$ be a finite list of real-valued variables called *auxiliary variables*.
- Let $r_1, \ldots, r_q$ be real-valued variables; they will be called the *results* of the computations.
- An *algorithm* $\mathcal{A}$ is a finite sequence of *instructions* $I_1, \ldots, I_N$ each of which has one of the following forms:
  - *assignment:* "$y \leftarrow y_1$" or "$y \leftarrow f_i(y_1, \ldots, y_{n_i})$";
  - *branching:* "*go to* I_i"; or "*if* $y_1 \odot y_2$, *then to* I_i *else go to* I_j", where $\odot$ is $=$, $\neq$, $<$, $>$, $\leq$, or $\geq$;
  - "*stop*".

10. Example

- Suppose that we have a copula

$$f_1(z_1, \ldots, z_n) = C(z_1, \ldots, z_n).$$

- Suppose also that we can use the values $v_{n+i} = F_i(x_i)$ as additional inputs.

- Then, the corresponding algorithm for computing the cdf has a running time of 1:

  I_1: $r_1 \leftarrow f_1(v_{n+1}, \ldots, v_{2n})$;

  I_2: stop.

11. What is a Computer Representation of a Multi-D Distribution

- By a *representation of an* n*-dimensional probability distribution*, we mean a tuple consisting of:
  - finitely many fixed functions $G_i(z_1, \ldots, z_{n_i})$, same for all distributions (such as $+$, $\cdot$, etc.);
  - finitely many functions $H_i(z_1, \ldots, z_{m_i})$ which may differ for different distributions; and
  - an algorithm (same for all distributions) that:
    - using the above functions and $2n$ inputs $x_1, \ldots, x_n, F_1(x_1), \ldots, F_n(x_n)$,
    - computes the values of the cdf $F(x_1, \ldots, x_n)$.
- *Examples of representations*:
  - original cdf one: $H_1(z_1, \ldots, z_n) = F(z_1, \ldots, z_n)$;
  - copula one: $H_1(z_1, \ldots, z_n) = C(z_1, \ldots, z_n)$.

12. A Representation Must Be Duplication-Free and Time-Efficient

- We say that a representation is *duplication-free* if no algorithm is possible that computes a marginal, given
  - the functions H_i representing the distribution and
  - the inputs $x_1, \ldots, x_n$.
- We say that a duplication-free representation is *time-efficient* if for each combination of inputs:
  - the running time of the corresponding algorithm does not exceed
  - the running time of any other duplication-free algorithm.

13. Computationally Efficient Representations

- Let $R = \langle H_1(z_1, \ldots, z_{m_1}), \ldots, H_k(z_1, \ldots, z_{m_k})\rangle$ and $R' = \langle H'_1(z_1, \ldots, z_{m'_1}), \ldots, H'_{k'}(z_1, \ldots, z_{m'_{k'}})\rangle$.
- We say that R if *more space-efficient* than R' if:
  - $k \le k'$ and
  - we can sort the value m_i and m'_i in such as way that $m_i \le m'_i$ for all $i \le k$.
- We say that a time-efficient duplication-free representation is *computationally efficient* if:
  - it is more space-efficient
  - than any other time-efficient duplication-free representation.
- **Main Result.** *The only computationally efficient duplication-free representation is the copula one.*
- This explains why copulas have been efficient.

14. Conclusions

- The need for representing multi-D distributions in a computer comes from the fact that:
  - to make decisions,
  - we need to be able to compute (and compare) the expected values of different utility functions.
- So:
  - from all possible computer representations of multi-D distributions,
  - we should select the ones for which the corresponding computations are the most efficient.
- We have shown that:
  - in situations where we already know the marginals,
  - copulas are indeed the most computationally efficient representation.

15. Possible Future Work

- In this paper, we have concentrated on computing the cumulative distribution function (cdf).
- This computation corresponds to *binary* utility functions, that take only two values $u^+ > u^-$.
- Such binary functions provide a good first approximation to the user's utilities and user's preferences.
- It is therefore desirable to find out,
  - for wider classes of utility functions,
  - which computer representations are the most computationally efficient for computing $E[u]$.
- Maybe copula-based computer representations will still be the most computationally efficient?

16. Acknowledgments

We acknowledge the partial support of:

- the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand;
- the National Science Foundation grants:
  - HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and
  - DUE-0926721.

17. Proof: Main Idea

- The copula representation is duplication-free and has running time $t = 1$.
- Thus, a time-efficient algorithm must have $t = 1$, i.e., it must have exactly one non-step instruction.
- We did not have time to compute any auxiliary values, so this instruction $r_1 \leftarrow f_1(y_1, \ldots, y_{n_1})$ uses inputs.
- Copula representation uses one function of n variables.
- Thus, a space-efficient repr. must have $n_1 \leq n$.
- If one of the inputs y_i is x_j, then:
  - by taking $y_i = 1$ or ∞, we would be able to compute the i-th marginal,
  - but we consider duplication-free representations.
- Thus, all inputs are marginals, so the rule is $r_1 \leftarrow H_1(F_1(x_1), \ldots, F_n(x_n))$, hence H_1 is a copula.