Why Copulas?

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

[1] Department of Computer Science, University of Texas at El Paso
500 W. University, El Paso, TX 79968, USA, vladik@utep.edu
[2] Department of Mathematical Sciences, New Mexico State University
Las Cruces, New Mexico 88003, USA, hunguyen@nmsu.edu
[3] Department of Economics, Chiang Mai University
Chiang Mai, Thailand, songsakecon@gmail.com
[4] University of Texas at El Paso, 500 W. University,
El Paso, TX 79968, USA, olgak@utep.edu

Abstract. A natural way to represent a 1-D probability distribution is to store its cumulative distribution function (cdf) $F(x) = \text{Prob}(X \le x)$. When several random variables $X_1, \ldots, X_n$ are independent, the corresponding cdfs $F_1(x_1)$, $\ldots$, $F_n(x_n)$ provide a complete description of their joint distribution. In practice, there is usually some dependence between the variables, so, in addition to the marginals $F_i(x_i)$, we also need to provide an additional information about the joint distribution of the given variables. It is possible to represent this joint distribution by a multi-D cdf $F(x_1, \ldots, x_n) = \text{Prob}(X_1 \le x_1 \,\&\, \ldots \,\&\, X_n \le x_n)$, but this will lead to duplication – since marginals can be reconstructed from the joint cdf – and duplication is a waste of computer space. It is therefore desirable to come up with a duplication-free representation which would still allow us to easily reconstruct $F(x_1, \ldots, x_n)$. In this paper, we prove that the only such representation is a representation in which marginals are supplements by a copula. This result explains why copulas have been successfully used in many applications of statistics.

1 How to Represent Probability Distributions: Formulation of the Problem

Probability distributions are ubiquitous. One of the main objectives of science and engineering is to predict the future state of the world – and to come up with decisions which lead to the most preferable future state.

These predictions are based on our knowledge of the current state of the world, and on our knowledge of how the state of the world changes with time. Our knowledge is usually approximate and incomplete. As a result, based on our current knowledge, we cannot predict the *exact* future state of the world, *several* future states are possible based on this knowledge. What we can predict is the set of possible states, and the *frequencies* with which, in similar situations, different future states will occur. In other words, what we can product is a *probability distribution* on the set of all possible future states.

This is how many predictions are made: weather predictions give us, e.g., a 60% chance of rain; economic predictions estimate the probability of different stock prices, etc.

Need to consider random variables. Information about the world comes from measurements. As a result of each measurement, we get the values of the corresponding physical quantities. Thus, a natural way to describe the state of the world is to list the values of the corresponding quantities $X_1, \ldots, X_n$.

From this viewpoint, the probability distribution on the set of all possible states means a probability distribution on the set of the corresponding tuples $X = (X_1, \ldots, X_n)$.

How to represent probability distributions: an important question. Due to ubiquity of probability distributions, it is important to select an appropriate computer representation of these distributions, a representation that would allow us to effectively come up with related decisions.

Thus, to come up with the best ways to represent a probability distribution, it is important to take into account how decisions are made.

How decisions are made: a reminder. In the idealized case, when we are able to exactly predict the consequences of each possible decision, decision making is straightforward: we select a decision for which the consequences are the best possible. For example:

- an investor should select the investment that results in the largest return,
- a medical doctor should select a medicine which leads to the fastest recovery of the patient, etc.

In reality, as we have mentioned, we can rarely predict the exact consequence of different decisions; we can, at best, predict the probabilities of different consequences of each decision. In such real-life setting, it is no longer easy to select an appropriate decision. For example:

- if we invest money in the US government bonds, we get a small guaranteed return;
- alternatively, if we invest into stocks, we may get much higher returns, but we may also end up with a loss.

Similarly:

- if we prescribe a well-established medicine, a patient will slowly recover;
- if instead we prescribe a stronger experimental medicine, the patient will probably recover much faster, but there is also a chance of negative side effects which may drastically delay the patient's recovery.

Researchers have analyzed such situations. The main result of the corresponding decision theory is that a consistent decision making under such probability uncertainty can be described as follows (see, e.g., [1, 3, 6]):

- 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.

Since we want a representation of a probability distribution that would make decision making as efficient as possible, we thus need to select a representation that would allow us to compute the expected values of different utility functions as efficiently as possible.

What is the most efficient representation of a 1-D probability distribution. How can we describe different utility functions? To answer this question, let us start by considering a simple example: the problem of getting from point A to point B.

In general, all else being equal, we would like to get from A to B as fast as possible. So, in the idealized case, if we knew the exact driving time, we should select the route that takes the shortest time. In practice, random delays are possible, so we need to take into account the cost of different delays.

In some cases – e.g., if we drive home after a long flight – a small increase of driving time leads to a small decrease in utility. However, in other cases – e.g., if we are driving to the airport to take a flight – a similar small delay can make us miss a flight and thus, the corresponding decrease in utility will be huge. In our analysis, we need to take into account both types of situations.

In the situations of the first type, utility $u(x)$ is a smooth function of the corresponding variable x. Usually, we can predict x with some accuracy, so all possible values x are located in a small vicinity of the predicted value x_0. In this vicinity, we can expand the dependence $u(x)$ in Taylor series and safely ignore higher order terms in this expansion:

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

The expected value of this expression can be thus computed as the linear combination of the corresponding moments:

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

Thus, to deal with situations of this type, it is sufficient to know the first few moments of the corresponding probability distribution.

In situations of the second type, we have a threshold x_t such that the utility is high for $x \le x_t$ and low for $x > x_t$. In comparison with the difference between high and low utilities, the differences between two high utility values (or, correspondingly, between two low utility values) can be safely ignored. Thus, we can simply say that $u = u^+$ for $x \le x_t$ and $u = u^- < u^+$ for $x > x_t$. In this case, the expected value of utility is equal to $E[u] = u^- + (u^+ - u^-) \cdot F(x_t)$, where $F(x_t) = \text{Prob}(x \le x_t)$ is the probability of not exceeding the threshold. So, to deal with situations of this type, we need to know the cdf $F(x)$.

In general, we need to know the cdf *and* the moments. Since the moments can be computed based on cdf, as $E[(x - x_0)^k] = \int (x - x_0)^k \, dF(x)$, it is thus sufficient to have a cdf.

How to represent multi-D probability distributions. In the multi-D cases, we similarly have two types of situations. For situations of the first type, when small changes in the values x_i lead to small changes in utility, it is sufficient to know the first few moments.

In the situations of the second type, we want all the values not to exceed appropriate thresholds. For example, we want a route in which the travel time does not exceed a certain pre-set quantity, and the overall cost of all the tolls does not exceed a certain value. To handle such situations, it is desirable to know the values of the corresponding multi-D cdf

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

Since the moments can be computed based on the cdf, it is thus sufficient to have a cdf.

Remaining problem. In some situations, we acquire all our knowledge about the probabilities in one step:

- we start "from scratch", with no knowledge at all,
- then we gain the information about the joint probability distribution.

In such 1-step situations, as we have just shown, the ideal representation of the corresponding probability distribution is by its cdf $F(x_1,\ldots,x_n)$.

In many practical situations, however, knowledge comes gradually. Usually, first, we are interested in the values of the first quantity, someone else may be interested in the values of the second quantity, etc. The resulting information is provided by the corresponding marginal distributions $F_i(x_i)$.

After that, we may get interested in the relation between these quantities $X_1,\ldots,X_n$. Thus, we would like to supplement the marginal distributions with an additional information that would enable us to reconstruct the multi-D cdf $F(x_1,\ldots,x_n)$.

In principle, we can store this multi-D cdf as the additional information. However, this is not the most efficient approach. Indeed, it is well known that each marginal distribution $F_i(x_i)$ can be reconstructed from the multi-D cdf, as

$$F_i(x_i)=F(+\infty,\ldots,+\infty,x_i,+\infty,\ldots,\infty)=\lim_{T\to\infty}F(T,\ldots,T,x_i,T,\ldots,T).$$

So, if we supplement the original marginals with the multi-D cdf, we thus store duplicate information, and duplication is a waste of computer memory.

It is therefore desirable to come up with an alternative representation, a representation that would avoid duplication, and that would still allow us to easily reconstruct the multi-D cdf.

What we do in this paper. In this paper, we prove that the only such representation is a representation in which marginals are supplements by a copula (see definition below). This result explains why copulas are successfully used in many applications of statistics [2, 4, 5].

The paper is structured as follows. In Section 2, we remind the reader what is a copula, and how copulas can be used to represent multi-D distributions. In

Section 3, we describe and prove the main result of this paper – that copulas are the only duplicate-free efficient representation of multi-D distributions.

2 Copulas: Brief Reminder

What is a copula. A copula corresponding to a multi-D distribution with cdf $F(x_1, \ldots, x_n)$ is a function $C(x_1, \ldots, x_n)$ for which

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

where $F_i(x_i)$ are the corresponding marginal distributions; see, e.g., [2, 4, 5].

How copulas can be used to represent multi-D distributions. A copula-related way to represent a multi-D distribution is to supplement the marginals $F_i(x_i)$ with the copula $C(x_1, \ldots, x_n)$.

The above formula then enables us to reconstruct the multi-D cdf $F(x_1, \ldots, x_n)$. This representation has no duplication, since for the same copula, we can have many different marginals.

3 Definitions and the Main Result

We want an algorithm for reconstructing $F(x_1, \ldots, x_n)$. We want to be able, given the marginals and the additional function(s) used for representing the distribution, to reconstruct the multi-D cdf $F(x_1, \ldots, x_n)$. This reconstruction has to be done by a computer *algorithm*.

An algorithm is 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 our computations, we can use inputs, we can use auxiliary variables, and we can use constants. In accordance with the IEEE 754 standard describing computations with real numbers, infinite values $-\infty$ and $+\infty$ can be used as well.

It is also possible to declare a variable as "undefined" (in IEEE 754 this is called "not a number", NaN for short). For each function or operation, if at least one of the inputs is undefined, the result is also undefined.

Let us give a formal definition of an algorithm.

Definition 1.

- *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 variable 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:*

- *an* assignment *instruction "$y \leftarrow y_1$" or "$y \leftarrow f_i(y_1, \ldots, y_{n_i})$", where:*
  - *y is one of the auxiliary variables or a result variable,*
  - *$f_i \in F$, and*
  - *each y_i is either an input, or an auxiliary variable, or a result, or a real number (including $-\infty$, $+\infty$, and NaN);*
- *an* unconditional branching instruction "go to I_i;
- *a* conditional branching *instruction* "if $y_1 \odot y_2$, then to I_i else go to I_j", *where:*
  - *each y_i is either an input, or an auxiliary variable, or the result, or a real number (including $-\infty$ and $+\infty$); and*
  - *$\odot$ is one of the symbols $=$, $\neq$, $<$, $>$, $\leq$, and $\geq$;*
- *or a* stopping *instruction* "stop"*.*

Definition 2. *The* result *of applying an algorithm $\mathcal{A}$ to the inputs $a_1, \ldots, a_m$ is defined as follows:*

- *in the beginning, we start with the given values of the inputs, all other variables are undefined;*
- *we then start with instruction I_1;*
- *on each instruction:*
  - *if this is an assignment instruction $y \leftarrow y_1$ or $y \leftarrow f_i(y_1, \ldots, y_{n_i})$, we assign, to the variable y, the new value y_1 or $f_i(y_1, \ldots, y_{n_i})$ and go to the next instruction;*
  - *if this is an unconditional branching instruction, we go to instruction I_i;*
  - *if this is a conditional branching instruction and both values y_1 and y_2 are defined, we check the condition $y_1 \odot y_2$ and, depending of whether this condition is satisfied, go to instruction I_i or to instruction I_j;*
  - *if this a conditional branching instruction, and at least one of the values y_i is undefined, we stop;*
  - *if this a stopping instruction, we stop.*

The values $r_1, \ldots, r_q$ at the moment when the algorithm stops are called the result *of applying the algorithm.*

Definition 3. *For every algorithm $\mathcal{A}$ and for each tuple of inputs $v_1, \ldots, v_m$, the number of instructions that the algorithm goes through before stopping is called the* running time *of $\mathcal{A}$ on $v_1, \ldots, v_m$.*

Examples. To illustrate the above definition, let us start with simple algorithms.

1°. The standard algorithm for computing the value $r_1 = v_1 \cdot (1 - v_1)$ requires the use of two arithmetic operations: subtraction $f_1(z_1, z_2) = z_1 - z_2$ and multiplication $f_2(z_1, z_2) = z_1 \cdot z_2$. Here, we can use a single auxiliary variable a_1. The corresponding instructions have the following form:

I_1: $a_1 \leftarrow f_1(1, v_1)$; this instruction computes $a_1 = 1 - v_1$;
I_2: $r_1 \leftarrow f_2(v_1, a_1)$; this instruction computes $r_1 = v_1 \cdot a_1 = v_1 \cdot (1 - v_1)$;

```
$I_3$: stop.
```

For all the inputs, this algorithm goes through two instructions before stopping, so its running time is 2.

2°. Computation of the absolute value $|v_1|$, i.e., v_1 if $v_1 \geq 0$ and $-v_1$ otherwise, requires that we use a unary minus operation $f_1(z_1) = -z_1$. The corresponding instructions have the following form:

```
$I_1$: if $v_1 \geq 0$, then go to $I_2$ else go to $I_4$;
$I_2$: $r_1 \leftarrow v_1$;
$I_3$: stop;
$I_4$: $r_1 \leftarrow f_1(v_1)$;
$I_5$: stop.
```

This algorithm also goes through two instructions before stopping, so its running time is also 2.

3°. Computation of $n! = 1 \cdot 2 \cdot \ldots n$ for a given natural number n requires:

- two arithmetic operations: addition $f_1(z_1, z_2) = z_1 + z_2$ and multiplication $f_2(z_1, z_2) = z_1 \cdot z_2$; and
- a loop, with an additional variable a_1 that takes the values 1, 2, ..., n.

The corresponding instructions have the following form:

```
$I_1$: $r_1 \leftarrow 1$;
$I_2$: $a_1 \leftarrow 1$;
$I_3$: if $a_1 \leq v_1$, then go to $I_4$ else go to $I_7$;
$I_4$: $r_1 \leftarrow f_2(r_1, a_1)$;
$I_5$: $a_1 \leftarrow f_1(a_1, 1)$;
$I_6$: go to $I_3$;
$I_7$: stop.
```

The running time of this algorithm depends on the input v_1.

- When $v_1 = 0$, we go through three instructions I_1, I_2, and I_3 before stopping, so the running time is 3.
- When $v_1 = 2$, we go through I_1, I_2, I_3, I_5, I_6, then again I_3, I_4, I_5, and I_6, and finally I_3 and stop. In this case, the running time is 11.

4°. If we already have the multi-D cdf as one of the basic functions $f_i(z_1, \ldots, z_n) = F(z_1, \ldots, z_n)$, then computing cdf for given inputs requires a single computational step:

```
$I_1$: $r_1 \leftarrow f_1(v_1, \ldots, v_n)$;
$I_2$: stop.
```

The running time of this algorithm is 1.

5°. Similarly, if we have a copula $f_1(z_1, \ldots, z_n) = C(z_1, \ldots, z_n)$, and we can use the values $v_{n+i} = F_i(x_i)$ as additional inputs, the corresponding algorithm for computing the cdf also has a running time of 1:

```
$I_1$: $r_1 \leftarrow f_1(v_{n+1}, \ldots, v_{2n})$;
$I_2$: stop.
```

Definition 3. *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, ..., x_n, $F_1(x_1)$, ..., $F_n(x_n)$, computes the values of the cdf $F(x_1, \ldots, x_n)$.*

Examples.

- In the original representation by a cdf, we have $H_1(z_1, \ldots, z_n) = F(z_1, \ldots, z_n)$.
- In the copula representation, we have $H_1(z_1, \ldots, z_n) = C(z_1, \ldots, z_n)$.

The corresponding algorithms for computing the cdf are described in the previous text.

Definition 4. *We say that a representation is* duplication-free *if no algorithm is possible that, given the functions H_i representing the distribution and the inputs $x_1, \ldots, x_n$, computes one of the marginals.*

Examples. The original representation by a cdf, when we have $H_1(z_1, \ldots, z_n) = F(z_1, \ldots, z_n)$, is not duplication-free, since we can compute, e.g., the marginal $F_1(v_1)$ by applying the following algorithm:

```
$I_1$: $r_1 \leftarrow H_1(v_1, +\infty, \ldots, +\infty)$;
$I_2$: stop.
```

On the other hand, the copula representation is duplication-free: indeed, for the same copula, we can have different marginals, and thus, it is not possible to compute the marginals based on the copula.

Definition 5. *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.*

Discussion. As we have mentioned earlier, in addition to an efficient use of computation time, it is also important to make sure that computer memory is used efficiently: this is why it makes sense to consider only duplication-free representations.

In general, we store the values of one of several functions of different number of variables. To store a function of m variables, we need to store, e.g., its values on the corresponding grid. If we use g different values of each of the coordinates, then we need to store the values of this function at g^m points, i.e., we need to store g^m real numbers. Thus, the smaller m, the more efficient we are. This leads to the following definition.

Definition 6.

- *We say that a representation $H_1(z_1, \ldots, z_{m_1})$, $\ldots$, $H_k(z_1, \ldots, z_{m_k})$ if* more space-efficient *than a representation $H'_1(z_1, \ldots, z_{m'_1})$, $\ldots$, $H'_{k'}(z_1, \ldots, z_{m'_{k'}})$ 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 is is more space-efficient than any other time-efficient duplication-free representation.*

Main Result. *The only computationally efficient duplication-free representation of multi-D probability distributions is the copula representation.*

Discussion. Thus, copulas are indeed the most efficient way of representing additional information about the multi-D distributions for which we already know the marginals. This theoretical result explains why copulas have been efficiently used in many applications.

Proof.

1°. By definition, a computationally efficient representation should be time-efficient. By definition of time efficiency, this means that for each combination of inputs, the running time of the corresponding algorithm should not exceed the running time of any other duplication-free algorithm.

We know that the copula representation is duplication-free and that its running time is 1 for all the inputs. Thus, for all the inputs, the running time of the computationally efficient algorithm should not exceed 1. Thus, this algorithm can have exactly one non-stop instruction.

2°. This instruction is our only chance to change the value of the output variable r_1, so this instruction must be of assignment type $r_1 \leftarrow f_1(y_1, \ldots, y_{n_1})$. Since we did not have time to compute the values of any auxiliary variables – this is our first and only instruction – the values $y_1, \ldots, y_{n_1}$ must be the original inputs.

3°. The function f_1 cannot be from the list of fixed functions, since otherwise

- we would get the same result for all possible probability distributions, and thus,
- we would not be able to compute the corresponding values of the cdf $F(x_1, \ldots, x_n)$, which are different for different distributions.

Thus, the function f_1 must be one of the functions H_i characterizing a distribution.

4°. This function $f_1 = H_i$ cannot have fewer than n inputs, because otherwise, some variable x_j will not be used in this computation. Thus, the list of functions

H_i used to describe a probability distribution must include at least one function of n variables.

5°. We are interested in a computationally efficient duplication-free representation. By definition, this means that this representation must be more space-efficient than any other time-efficient duplication-free representation. We know one time-efficient duplication-free representation – it is the copula representation, in which we use a single function H_1 of n variables.

The fact that our representation is more space-efficient than this one means that it uses only one function, and this must be a function of n or fewer variables. We have already shown that we cannot have a function of fewer than n variables, so we must have a function of exactly n variables.

6°. The result $F(x_1, \ldots, x_n)$ of applying this function of n variables must depend on all n variables $x_1, \ldots, x_n$. Thus, for each of these variables x_i, either this same value x_i or the value $F_i(x_i)$ must be among its inputs.

7°. If one of the inputs is x_i, i.e., if the corresponding instruction has the form

I_1: $r_1 \leftarrow H_1(y_1, \ldots, y_{i-1}, x_i, y_{i+1}, \ldots, y_n)$;

where each y_i is either x_i or $F_i(x_i)$, then we will be able to compute the corresponding marginal by using the instruction

I_1: $r_1 \leftarrow H_1(Y_1, \ldots, Y_{i-1}, x_i, Y_{i+1}, \ldots, Y_n)$;

where $Y_i = +\infty$ when $y_i = x_i$ and $Y_i = 1$ when $y_i = F_i(x_i)$. Since we assumed that our scheme is duplication-free, this means that such a case is not possible, and thus, all the inputs to the function H_1 are not the values x_i, but the values of the marginals. Thus, the corresponding instruction has the form

I_1: $r_1 \leftarrow H_1(F_1(x_1), \ldots, F_n(x_n))$;

The result of this computation should be the multi-D cdf, so we should have

$$F(x_1, \ldots, x_n) = H_1(F_1(x_1), \ldots, F_n(x_n))$$

for all possible values $x_1, \ldots, x_n$.

This is exactly the definition of the copula, so we indeed conclude that every computationally efficient representation of a multi-D probability distribution is the copula representation. The main result is proven.

Acknowledgments

We acknowledge the partial support of the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand. This work was also supported in part by the National Science Foundation grants HRD-0734825 and HRD-1242122 (Cyber-ShARE Center of Excellence) and DUE-0926721.

References

1. P. C. Fishburn, *Utility Theory for Decision Making*, John Wiley & Sons Inc., New York, 1969.
2. P. Jaworski, F. Durante, W. K. Härdle, and T. Ruchlik (eds.), *Copula Theory and Its Applications*, Springer Verlag, Berlin, Heidelberg, New York, 2010.
3. R. D. Luce and R. Raiffa, *Games and Decisions: Introduction and Critical Survey*, Dover, New York, 1989.
4. A. J. McNeil, R. Frey, and P. Embrechts, *Quantitative Risk Management: Concepts, Techniques, Tools*, Princeton University Press, Princeton, New Jersey, 2005.
5. R. B. Nelsen, *An Introduction to Copulas*, Springer Verlag, Berlin, Heidelberg, New York, 1999.
6. H. Raiffa, *Decision Analysis*, Addison-Wesley, Reading, Massachusetts, 1970.