Vine copula structure learning via Monte Carlo tree search

Bo Chang
Department of Statistics
University of British Columbia

Shenyi Pan
Department of Statistics
University of British Columbia

Harry Joe
Department of Statistics
University of British Columbia

Abstract

Monte Carlo tree search (MCTS) has been widely adopted in various game and planning problems. It can efficiently explore a search space with guided random sampling. In statistics, vine copulas are flexible multivariate dependence models that adopt vine structures, which are based on a hierarchy of trees to express conditional dependence, and bivariate copulas on the edges of the trees. The vine structure learning problem has been challenging due to the large search space. To tackle this problem, we propose a novel approach to learning vine structures using MCTS. The proposed method has significantly better performance over the existing methods under various experimental setups.

1 INTRODUCTION

Monte Carlo tree search (MCTS) is a search and planning framework for finding optimal decisions by taking random samples in the decision space (Browne et al., 2012). The key idea of MCTS is first to construct a search tree of states which are evaluated by fast Monte Carlo simulations and then selectively grow the tree (Coulom, 2006). Multi-armed bandit algorithms such as the upper confidence bounds for trees (UCT) can be employed to balance between exploration and exploitation (Kocsis and Szepesvári, 2006). As one of the most important methods in artificial intelligence, MCTS has been widely applied in various game and planning problems, including chess, shogi, Go, real-time video games, and even games with incomplete information such as poker. In March 2016, AlphaGo, which combines MCTS with deep neural networks, became the first computer Go program to beat a 9-dan professional without handicaps (Silver et al., 2016). This is regarded as a major milestone in artificial intelligence research.

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89. Copyright 2019 by the author(s).

In probability theory and statistics, a copula is a multivariate probability distribution with uniform $(0, 1)$ univariate margins. Via Sklar's theorem (Sklar, 1959), any multivariate distribution can be decomposed in terms of univariate marginal distribution functions and a copula which describes the dependence structure among the variables. There exist many parametric copula families, such as Gaussian and Student t-copulas. Copulas are widely applied in areas that require non-Gaussian multivariate or high-dimensional statistical inference. Joe (2014) includes a detailed introduction to copula theory, models and applications.

A vine is a graphical object represented by a sequence of connected trees. In a vine copula model, vine graphs are adopted to specify the dependence structure, and bivariate copulas are used as the basic building blocks on the edges of vines. Vine copulas have been proven to be a flexible tool in high-dimensional (non-Gaussian) dependence modeling, and have been applied to various fields including finance (Dissmann et al., 2013), longitudinal studies (Cooke et al., 2015), and spatial statistics (Krupskii et al., 2018).

Recently, copulas and vine copulas have gained popularity in machine learning and artificial intelligence research. Elidan (2013) summarizes some applications of copula-based constructions in machine learning. Tran et al. (2015) use vine copulas to augment the mean-field factorization in variational inference. Ozdemir et al. (2017) use copulas to model the joint density of probability scores, leading to a new classifier fusion approach. Wieczorek et al. (2018) apply a Gaussian copula to the problem of feature disentanglement in the latent space by restoring the invariance properties of the information bottleneck method.

The structure learning of the vine is computationally intractable in general. There are a large number of possible vine structures which result in a huge search space for a high-dimensional dataset if one would like to find the optimal one. Specifically, according to Cayley's formula, one can construct d^{d-2} different trees with d nodes. With this result, Kurowicka and Joe (2011) further show that there are in total $2^{(d-3)(d-2)}(d!/2)$ different vine structures considering all levels for a dataset with d variables. This makes vine structure searching and learning a challenging problem. Previous work has been mainly centered around greedy algorithms which follow the heuristics of joining variables with stronger dependence in low-level trees and making the locally optimal choice at each tree level conditional on previous trees. However, it does not in general produce a solution that is near the global optimum of the vine structure learning problem.

Because the construction of a truncated vine is inherently sequential, we formulate the vine structure learning problem as a sequential decision making process in this work. A search tree thus arises, where the root node is "empty" and the terminal leaf nodes are valid vine structures. Although the height and branching factor of the search tree might be large, MCTS can be adopted to search through it efficiently. Specifically, we adapt the existing upper confidence bounds for trees (UCT) algorithm for vine structure learning and incorporate tree policy enhancements including first play urgency, progressive bias, and efficient transposition handling. The adapted UCT is called the vine UCT, under the guidance of which, the tree policy strikes a balance between exploration and exploitation. The proposed method is applied to datasets from various disciplines and compared against existing methods. All the experiments suggest that the proposed method outperforms existing methods.

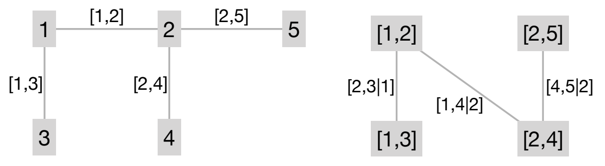
The remainder of the paper is organized as follows. Vines and vine structure learning are introduced and defined in Section 2. We give a brief overview of Monte Carlo tree search (MCTS) in Section 3. The proposed MCTS-based vine structure learning method is described in Section 4, and is evaluated on various datasets in Section 5. Section 6 provides concluding remarks.

2 VINE STRUCTURE LEARNING

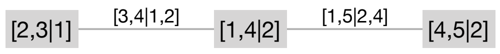
2.1 Vine Graphical Model

A vine is a nested set of trees where the edges in the first tree are the nodes of the second tree, the edges of the second tree are the nodes of the third tree, etc. Vines are useful in specifying the dependence structure for general multivariate distributions on d variables.

The first tree in a vine represents d variables as nodes and the bivariate dependence of $d-1$ pairs of variables as edges. The second tree describes the conditional dependence of $d-2$ pairs of variables conditioning on another variable; nodes are the edges in tree 1, and a pair of nodes could be connected if there is a common variable in the pair. The third tree describes the conditional dependence of $d-3$ pairs of variables conditioning on two other variables; nodes are the edges in tree 2, and a pair of nodes could be connected if there are two common conditioning variables in the pair. This continues until tree $d-1$ has only one edge that describes the conditional dependence of two variables conditioning on the remaining $d-2$ variables.



(a) Level 1 tree T_1. (b) Level 2 tree T_2.



(c) Level 3 tree T_3.

Figure 1: An example of a 3-truncated vine with $d = 5$.

For a concrete example, as shown in Figure 1, consider $d = 5$ variables labeled as $1, 2, 3, 4, 5$. Suppose tree 1 has edges $[1, 2]$, $[1, 3]$, $[2, 4]$, $[2, 5]$ where $[1, 2]$ is an edge connecting variables 1 and 2, etc. Possible edges for tree 2 are $[2, 3|1]$, $[1, 4|2]$, $[4, 5|2]$ where $[2, 3|1]$ connects $[1, 2]$ and $[1, 3]$ (edges of tree 1 are nodes in tree 2, and these two nodes have the variable 1 in common). Possible edges for tree 3 are $[3, 4|1, 2]$, $[1, 5|2, 4]$ where $[1, 5|2, 4]$ connects $[1, 4|2]$ and $[4, 5|2]$ (edges of tree 2 are nodes in tree 3, and these two nodes have the variables $2, 4$ in common). Note that the possible edges in a tree depend on but are not uniquely determined by the edges of the previous trees. For example, $[2, 3|1]$, $[1, 4|2]$, $[1, 5|2]$ is another possible set of edges for tree 2 in Figure 1. Therefore, one needs to decide which configuration to adopt when building trees for a new level. The requirement that *two connected nodes must have two distinct variables and the remaining variables in common* is called the **proximity condition**.

A formal definition is given as follows.

Definition 1 *(Vine) V is a vine on d variables if*

1. $V = (T_1, \dots, T_{d-1})$;
2. *T_1 is a tree with nodes $N(T_1) = \{1, 2, \dots, d\}$, and edges $E(T_1)$. For $\ell > 1$, T_ℓ is a tree with nodes $N(T_\ell) = E(T_{\ell-1})$;*
3. *(proximity condition) For $\ell = 2, \dots, d-1$, for $\{n_1, n_2\} \in E(T_\ell)$, $\#(n_1 \triangle n_2) = 2$, where $\triangle$ denotes symmetric difference and $\#$ denotes cardi-*

nality.

2.2 From Vine to Multivariate Distributions

To get a multivariate distribution from a vine, bivariate distributions are assigned to the edges of tree 1 and bivariate conditional distributions are assigned to the edges of trees $2, \ldots, d-1$. If the bivariate distributions on the edges are all bivariate Gaussian, each edge can be characterized by a correlation parameter ρ, which can be interpreted as a partial correlation for trees 2 to $d-1$. The partial correlation parameters can be computed in closed-form from the empirical correlation matrix $\mathbf{R}$. See the supplementary materials for a more detailed introduction to the representations of a multivariate Gaussian distribution through vines.

2.3 Truncated Vine

There are $d(d-1)/2 = O(d^2)$ edges in a complete vine graph, and at least $d(d-1)/2$ parameters for a vine copula with a parametric bivariate copula family on each edge. Great computational effort is required for parameter estimation in high-dimensional cases. Truncated vines are useful for representing the dependence of d variables in a parsimonious way. A **truncated vine** with $1 \le t < d-1$ trees, or a t-truncated vine, assumes that the most important dependencies are captured by the first t trees $V_t = (T_1, \ldots, T_t)$ in a vine and the remaining trees can be represented with conditional independence of two variables given the conditioning variables. In the Gaussian case, this is equivalent to assigning partial correlations of 0 to the edges of the remaining $d-t-1$ trees. By truncation, the number of parameters in a vine copula is reduced from $O(d^2)$ to $O(d)$, if t is constant as d increases.

The most parsimonious vine structure is a 1-truncated vine with one tree that connects $d-1$ pairs of variables. This is a valid structure (called a Markov tree) if variables not directly connected are conditionally independent given the variables in the tree path that connect them. However, a Markov tree can seldom summarize the dependence well in d variables. As an improvement, the truncated vine ($t \ge 2$) adds some layers of conditional dependence on top of a Markov tree until conditional independence relations from high-order trees are approximately valid.

2.4 Objective Function

Kurowicka and Cooke (2003) show that the log-determinant of the empirical correlation matrix $\mathbf{R}$ is $\log\det(\mathbf{R}) = \sum_{e\in E(V)} \log(1-\rho_e^2)$ for any vine V, with $\{\rho_e\}$ being the set of correlations and partial correlations on the edges of the vine. Assuming all the bivariate copulas are Gaussian, $\log\det(\mathbf{R})$ is also linearly related to the negative log-likelihood of the vine copula. The best t-truncated partial correlation vine to approximate the correlation matrix is such that $\sum_{e\in E(V_t)} \log(1-\rho_e^2)$ is close to $\log\det(\mathbf{R})$. This implies that one wants a truncated vine such that ρ_e^2 are large in the first t trees and small in the remaining trees.

Formally, the goal of the vine structure learning problem is to find a t-truncated vine that maximizes the objective function, or log-likelihood function

$$L_t(V) = \sum_{i=1}^{t} \sum_{e\in E(T_i)} -\log(1-\rho_e^2), \qquad (1)$$

where t is a pre-defined truncation level, and ρ_e is the partial correlation corresponding to edge e in the vine. Since $\rho_e^2 \in (0,1)$, $L_t(V)$ is monotonically increasing with respect to t. Furthermore,

$$L_{d-1}(V) = -\log\det(\mathbf{R}), \quad \forall V. \qquad (2)$$

In other words, all the $(d-1)$-truncated vines achieves the same objective function.

2.5 Existing Methods

A few existing methods attempt to solve the vine structure learning problem. The most direct way is to enumerate and compare all possible vine structures in a brute-force fashion. However, as mentioned earlier, there are $2^{(d-3)(d-2)}(d!/2)$ different vine structures in d variables, so this makes brute-force search only feasible for $d \le 8$ due to the exponentially increasing number of possible vine structures.

As an alternative, Dissmann et al. (2013) propose a method based on the maximum spanning tree (MST) algorithms with different choices for edge weights that reflect the strength of the dependence between pairs of variables. For multivariate Gaussian case, a good choice of edge weight in the trees is weight $-\log(1-\rho_e^2)$ for edge e; this is used in Section 6.17 of Joe (2014). The trees, T_1 to T_t, of the vine are sequentially constructed by maximizing the sum of the edge weights at each tree level. Such an MST can be obtained using the algorithm by Prim (1957). Dissmann's algorithm is a greedy algorithm: the construction of T_{i+1} is based on the locally optimal choice given T_i. It does not, in general, produce a globally optimal solution.

Inspired by genetic algorithms, Brechmann and Joe (2015) propose methods to explore the search space of truncated vines effectively. At each tree level, it considers not only the MST but also neighbors of the MST. In general, the results generated by this algorithm outperform the greedy algorithm.

More recently, Kraus and Czado (2017) propose a method that selects simplified vine copulas for which the simplifying assumption is violated as little as possible. They adopt the constant conditional correlation test and use the corresponding p-values to define edge scores; this is different from our objective to maximize the likelihood and find the optimal vine structure under the Gaussian copula assumption. Further, this method combines the vine structure learning with copula selection, which is beyond the scope of this paper. Therefore, this method is not directly comparable with our proposed method.

3 MONTE CARLO TREE SEARCH

3.1 Algorithm

Monte Carlo tree search (MCTS) is a method for trying to find the optimal sequence of decisions in a given domain by taking random samples in the decision space and building a search tree accordingly. It has been widely used in domains that can be represented as trees of sequential decisions, such as games and planning problems.

The basic MCTS algorithm is conceptually simple: it iteratively builds a search tree until a predefined computational budget is reached. The search tree is initialized to a root node v_0 with an initial state s_0. In each iteration, four steps are applied:

1. *Selection*: Starting at the root node v_0, a **tree policy** is recursively applied to descend from the root node to a node with at least one unvisited child node.
2. *Expansion*: An unvisited child node v_ℓ is added to expand the tree.
3. *Simulation*: A simulation is run from the new node v_ℓ according to a **default policy** to produce an outcome. The default policy is often a uniformly random sequence of actions applied to the state of v_ℓ until a terminal state is reached.
4. *Backpropagation*: The simulation result Δ, which is the outcome of the terminal state, is backpropagated through the selected nodes to update their statistics.

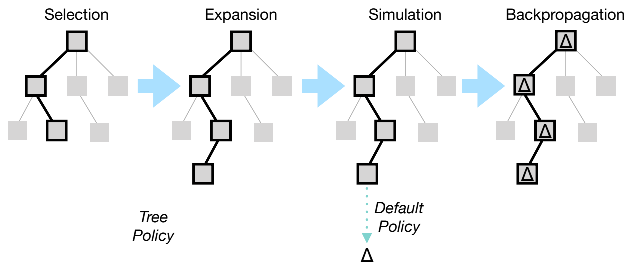


Figure 2: One iteration of the general MCTS algorithm.

The final result is a sequence of actions that lead to the best outcome starting from the root node v_0. One iteration of the MCTS algorithm is shown in Figure 2.

3.2 Upper Confidence Bounds for Trees (UCT)

The choice of tree policy is crucial: it attempts to balance between exploration (look in areas that have not been well sampled yet) and exploitation (look in areas which appear to be promising). In this section, we describe a popular tree policy in the MCTS family, the upper confidence bound for trees (UCT).

UCT is based on the principle of optimism in the face of uncertainty. With UCT, in the selection and expansion step a child node j is selected to maximize

$$\mathrm{UCT}(j) = \bar{x}_j + \kappa \sqrt{\frac{2 \log n}{n_j}}, \tag{3}$$

where

- $\bar{x}_j$ is the average outcome from child node j;
- n is the number of times the parent node has been visited;
- n_j is the number of times child j has been visited;
- $\kappa > 0$ is a constant.

If more than one child node has the same maximal value, the tie is usually broken randomly. Since $n_j = 0$ yields a UCT value of ∞, previously unvisited children are assigned the largest possible value, which corresponds to the expansion step in the MCTS algorithm. In the backpropagation step, $\bar{x}_j$ and n_j are updated accordingly.

4 PROPOSED METHOD

4.1 Vine Structure Learning as a Sequential Decision Making Problem

In this section, we introduce our proposed method of using MCTS to find the optimal structure of a t-truncated vine. A t-truncated vine can be represented by a sequence of edges in the sequence of trees $T_1, \ldots, T_t$. Given the discrete and hierarchical nature, the construction of a t-truncated vine can be regarded as a sequential decision making problem:

1. The initial state has no edge; T_i are empty for $i \in \{1, \ldots, d-1\}$.
2. Starting from level $i = 1$, add one edge to T_i at each step according to the tree policy. The candi-

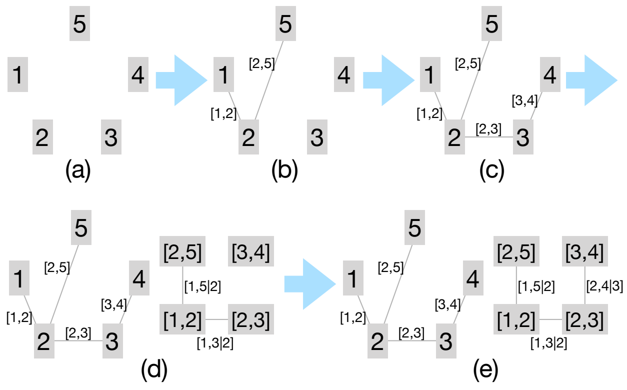


Figure 3: Vine structure learning as a sequential decision problem.

date edges are chosen so that T_i has only one connected and acyclic component. For levels $i > 1$, the candidate edges also need to satisfy the proximity condition.

3. If T_i is connected and $i < t$, go to step 2 and start adding edges to T_{i+1}.

Figure 3 shows an example of vine structure learning with dimension $d = 5$ and truncation level $t = 2$. Each subfigure represents a state in the search tree. **(a)** The initial state is an empty graph with 5 nodes. **(b)** After two steps, two edges $[1, 2]$ and $[2, 5]$ are added to the graph according to the tree policy. Note that given this state, $[1, 5]$ cannot be added otherwise a cycle would form. **(c)** Two more edges $[2, 3]$ and $[3, 4]$ are added and T_1 becomes connected. **(d)** We have started adding edges to T_2; $[1, 5|2]$ and $[1, 3|2]$ have been sequentially added. Given this state, edges $[2, 5]$–$[3, 4]$ and $[1, 2]$–$[3, 4]$ do not satisfy the proximity condition, and edge $[2, 3]$–$[2, 5]$ does not satisfy the acyclic condition. Therefore, the only edge that can be further added is $[2, 3]$–$[3, 4]$, namely $[2, 4|3]$. **(e)** All the edges have been added and this is a terminal state. The tree construction resembles Prim's algorithm (Prim, 1957), which ensures that at each time step there exists at most one connected component in each tree.

A default policy determines how to move from a non-terminal node to a terminal node in the search tree. In our application, it specifies how to "complete" the truncated vine given an incomplete one. We consider two options:

1. **Uniformly random**: given an incomplete truncated vine, an edge is randomly selected from all the eligible edges, that is the candidate edges that satisfy the acyclic and proximity conditions.
2. **Maximum spanning tree**: similar to Dissmann's Algorithm (Dissmann et al., 2013) introduced in Section 2.5, given an incomplete truncated vine, maximum spanning trees are constructed sequentially.

The disadvantage of maximum spanning tree default policy is that it greedily expands an incomplete vine, and this might lead to insufficient exploration and lock onto a set of suboptimal actions. Therefore, we use the uniformly random default policy in the proposed method to better explore the search space.

4.2 Vine UCT

The UCT algorithm is used as the tree policy in various sequential decision making domains. However, in order to apply the original UCT algorithm to the vine structure learning problem, a few adaptations are needed.

Scaling Constant MCTS algorithms are commonly adopted in artificial intelligent game playing problems, where the outcome is often binary: win or lose, represented by 1 and 0 respectively. As a result, the average outcome $\bar{x}_j$ in Equation 3 is always in the range of $[0, 1]$.

In our application, the objective function $L_t(V)$ is in the range of $(0, -\log\det(\mathbf{R})]$. We need to adjust the scaling constant κ in Equation 3 accordingly so that the exploration and exploitation terms are on the same scale. A natural choice of κ is $-\log\det(\mathbf{R})$.

First Play Urgency In the original UCT algorithm, the selection step stops whenever a node has an unvisited child node. For problems with large branching factors or height in the search tree, the tree will not grow deeper unless all the child nodes are visited. For vine structure learning, the height of the search tree is in the order of $O(d^2)$, and the branching factor is also large. Therefore, exploitation will rarely occur deep in the tree according to the original UCT algorithm. First play urgency (FPU) is a modification proposed by Gelly and Wang (2006) to address this issue. It assigns a fixed value of λ_{FPU} to score the unvisited nodes and uses the original UCT formula to score the visited nodes. By doing so, the score of an unvisited node is no longer infinite, and this encourages early exploitation.

Progressive Bias When a node has been visited only a few times, its statistics are not reliable. Progressive bias is a technique of adding domain specific heuristic knowledge to MCTS (Chaslot et al., 2008). In artificial intelligence game playing problems, many games already have strong heuristic knowledge.

A general form of progressive bias for node j is $H_j/(n_j+1)$, where H_j is a heuristic value and n_j is the number of visits for this node. This term is added to the UCT formula to encourage exploitation of nodes with larger heuristic values. As the number of visits n_j increases, the effect of progressive bias decreases.

In our application, given the objective function in Equation 1, H_j can be chosen as $H_j = -\log(1-\rho_{e_j}^2)$, since the objective function is the summation of H_j over all the edges in a truncated vine. Here e_j is the edge added by node j and ρ_{e_j} is the corresponding (partial) correlation parameter. A tuning parameter λ_{PB} controls the strength of progressive bias. When λ_{PB} is sufficiently large, the tree policy is solely controlled by the heuristic value, and the MCTS algorithm coincides with Dissmann's algorithm.

Transpositions The formulation of vine structure learning as a sequential decision making process has a potential problem: the same states can be reached via different paths in the search tree. This is usually referred to as transpositions.

Figure 4 shows the search tree corresponding to a 1-truncated vine with $d = 3$. It is obvious that there are only three unique 1-truncated vines in this case summarized as 1–2–3, 1–3–2, 2–1–3. However, the search tree contains six terminal nodes; for each unique vine structure, there exist two distinct paths leading to it. For example, the two paths $[1,2],[2,3]$ and $[2,3],[1,2]$ both result in the same vine structure 1–2–3.

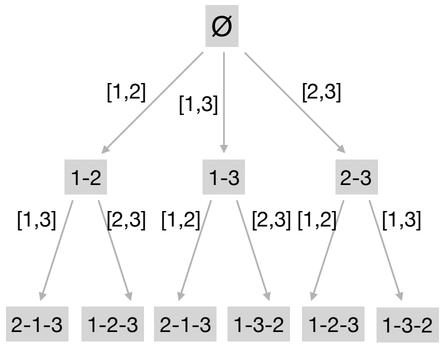


Figure 4: The search tree corresponding to a 1-truncated vine with $d = 3$.

Transpositions cause inefficiency because the statistics of the same state are scattered across different nodes. Transposition tables are the usual choice to tackle this problem; they store information about states and share the statistics to subsequent occurrences of the same state during the search. A transposition table is usually implemented as a hash table of the unique vine states. On encountering a new vine state, the algorithm checks the table to see whether the state has already been analyzed; this can be done quickly, in expected constant time. If the table contains the statistics that were previously assigned to this state, the statistics are used directly. Otherwise, this new state is entered into the hash table.

It is relatively straightforward to apply transposition tables in the selection steps of MCTS. Childs et al. (2008) further discuss the use of transposition tables in the backpropagation steps. Specifically, we adopt the UCT2 algorithm from that paper. Compared with the original UCT formula in Equation 3, there are two modifications: (1) a transposition table is used to share statistics of the same vine state; (2) the algorithm keeps track of the number of visits of both nodes and edges in the search tree. For a parent node p and its child node j, the UCT2 value is given by

$$\text{UCT2}(p,j) = \bar{x}_j + \kappa\sqrt{\frac{2\log n_p}{n_{(p,j)}}}, \tag{4}$$

where $\bar{x}_j$ is retrieved from the transposition table, n_p is the number of visits of node p and $n_{(p,j)}$ is the number of visits of edge (p,j). Note that if $n_{(p,j)}$ is replaced with n_j, the value of the parent node might converge to an incorrect value.

Vine UCT Combining all the above adaptations, the resulting UCT is called the vine UCT. For a parent node p and its child node j, the vine UCT value is

$$\begin{aligned}\text{VUCT}(p,j) = \bar{x}_j - \log\det(\mathbf{R})\Bigg[\lambda_{\text{PB}}\cdot\frac{H_j}{n_j+1} \\ + \min\Bigg\{\sqrt{\frac{2\log n_p}{n_{(p,j)}}}, \lambda_{\text{FPU}}\Bigg\}\Bigg],\end{aligned} \tag{5}$$

where

- λ_{FPU} and λ_{PB} are the tuning parameters;
- $H_j = -\log(1-\rho_{e_j}^2)$ is the contribution to the objective function by the newly added edge e_j in child j;
- n_p and n_j are the numbers of visits of parent node p and child node j;
- $n_{(p,j)}$ is the number of visits of edge (p,j);
- $\bar{x}_j$ is the average outcome from child node j retrieved from the transposition table.

In summary, the input of our method is a correlation matrix $\mathbf{R}$ calculated from a multivariate dataset. The MTCS algorithm is applied, using vine UCT as the tree policy and uniformly random default policy. In every MCTS iteration, the default policy leads to a terminal t-truncated vine. Through the MCTS iterations, we keep the vine with the largest value of the objective function, and it is returned as the output.

5 EXPERIMENTS

5.1 Datasets

To assess the effectiveness of our proposed method, we consider four datasets from various fields. Two of them (Concrete and Abalone) are small datasets with $d = 8$ so that it is feasible to run the brute-force algorithm. The other two have more variables (Glioblastoma Tumors (GBM) and Deutscher Aktien Index (DAX)). Subsets of variables are randomly selected from these two larger datasets to evaluate the performance of the proposed method. A detailed introduction to each dataset is included in the supplementary materials.

5.2 Performance Metric

It is a common question whether an empirical correlation matrix $\mathbf{R}$ is well approximated by a model. A likelihood-ratio test is often used to assess the goodness-of-fit of a structural model. The **comparative fit index** (CFI) is a fit index that takes into account the likelihood-ratio as well as the number of parameters in the model (Bentler, 1990; Brechmann and Joe, 2015).

The test statistic of the likelihood-ratio test is given by

$$D_t = n[-L_t(V) - \log\det(\mathbf{R})], \tag{6}$$

where L_t is the objective function defined in Equation 1. If the model is completely unstructured (the saturated model), then $D_t = 0$. On the other hand, if the model assumes that all variables are uncorrelated, then $D_0 := -n \log\det(\mathbf{R})$. Reasonable models should lie somewhere in between these two extreme cases. Therefore, D_t can be viewed as a discrepancy measure.

For a t-truncated vine, its degree of freedom (or $d(d-1)/2$ minus the number of model parameters) is

$$\nu_t = \frac{d(d-1)}{2} - \frac{t(2d-t-1)}{2} = \frac{(d-t)(d-t-1)}{2}. \tag{7}$$

In particular, $\nu_0 = d(d-1)/2$ is the case of complete independence.

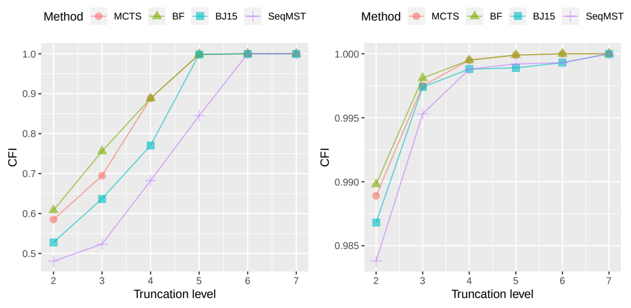
The CFI of a t-truncated vine is defined as

$$\mathrm{CFI}_t := 1 - \frac{\max(0, D_t - \nu_t)}{\max(0, D_0 - \nu_0, D_t - \nu_t)}, \tag{8}$$

which ranges between 0 and 1. Higher CFI values correspond to better fit. CFI can be used to find an optimal truncation level given a predefined goodness-of-fit level. Formally, the optimal truncation level is given by

$$t^*_\alpha = \min\{t \in \{0, \dots, d-1\} : \mathrm{CFI}_t \geq 1 - \alpha\}, \tag{9}$$

where $\alpha \in (0, 1)$. Commonly used α values include 0.01 and 0.05.



(a) Concrete dataset. (b) Abalone dataset.

Figure 5: CFI vs truncation level t.

5.3 Results

For all the experiments, 5000 MCTS iterations are performed; hyperparameters are set to $\lambda_{\mathrm{FPU}} = 1$ and $\lambda_{\mathrm{PB}} = 0.1$. The following results show that for datasets with various dimensions d and truncation levels t, this set of hyperparameters consistently gives decent results. This indicates that our method is robust under different settings and does not require much hyperparameter tuning. The algorithm is implemented in Python and code will be made publicly available.

The comparisons have been made on correlation matrices of actual datasets. It is possible to construct synthetic structural correlation matrices for which SeqMST performs much worse than the examples in this section.

Concrete and Abalone datasets

On both datasets, the proposed method (MCTS) is compared with the following baseline methods: (1) **BF**: the brute-force search; (2) **SeqMST**: Dissmann's algorithm (Dissmann et al., 2013); (3) **BJ15**: method proposed by Brechmann and Joe (2015). See Section 2.5 for details about the baseline methods.

Figure 5 shows the CFI for various truncation levels t. Higher CFI values indicate better fit. Note that 1-truncated vines are omitted from the figures because the optimal 1-truncated vine can be found by an MST algorithm.

For both datasets, the performance of MCTS is better than SeqMST and BJ15 for all truncation levels. Notably, MCTS has the same performance as BF for $t = 4, 5$ and 6, which indicates that our method can find the best truncated vine for those truncation levels for both datasets.

GBM and DAX datasets

Figure 6 shows the CFI for different truncation levels t and various dimensions d on the GBM dataset.

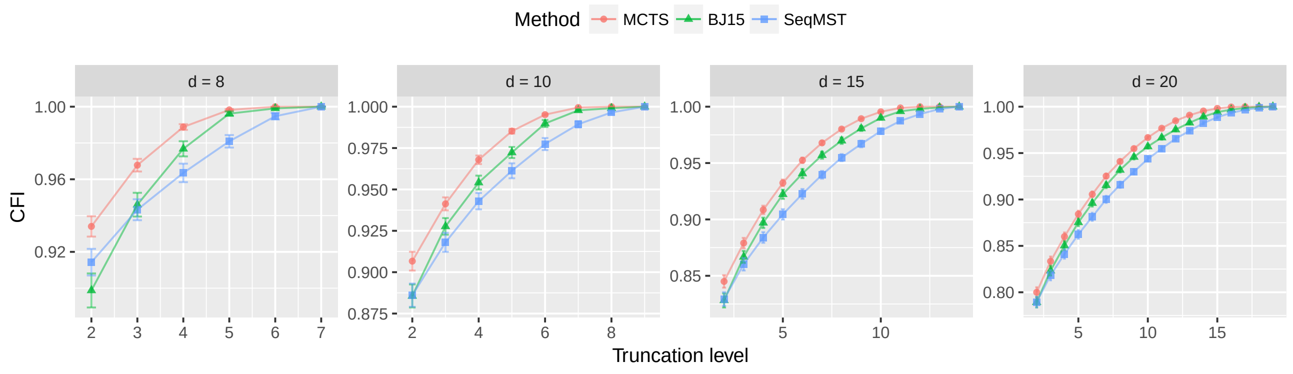
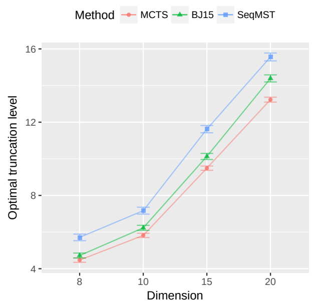


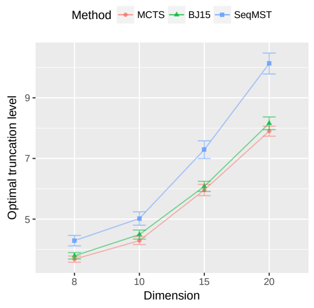
Figure 6: GBM dataset: CFI vs truncation level t.

Since 100 subsets of variables are randomly sampled for each $d = 8, 10, 15$ and 20, confidence intervals can be obtained. Note that the brute-force search is no longer feasible in this experiment. MCTS outperforms both BJ15 and SeqMST on all the combinations of truncation level t and dimension d. Especially when the truncation level t is small, MCTS is significantly better than the existing methods.

Another way to demonstrate the performance is to compare the optimal truncation level t^*_α defined in Equation 9. It gives the minimal truncation level that reaches a CFI level of $1 - \alpha$. The lower t^*_α, the more dependence information captured by the vine structure. Figure 7 shows the the optimal truncation level $t^*_{\alpha=0.01}$ for both GBM and DAX datasets. For the GBM dataset with $d = 20$, MCTS selects a vine with 2.4 fewer trees over SeqMST and 1.2 fewer trees over BJ15 on average. The performance for larger values of d such as $d = 30$ is similar.



(a) GBM dataset.



(b) DAX dataset.

Figure 7: Optimal truncation level $t^*_{\alpha=0.01}$ vs dimension d.

5.4 Computational Efficiency

To evaluate the computational efficiency of our proposed method, we measure the running time of each algorithm on the Concrete dataset for truncation level 4. The experiment is repeated 10 times for each algorithm and the wall clock time is listed in Table 1.

	MCTS	**SeqMST**	**BJ15**	**BF**
Time (sec)	37.6	6.0	26.9	52.6

Table 1: The wall clock time running each algorithm on the Concrete dataset for truncation level 4.

The proposed MCTS method is slower than SeqMST but has comparable performance as BJ15. However, the SeqMST algorithm is efficiently implemented in R and C, while the implementation of our method is purely in Python. There is room for improvement in computational efficiency. We will explore ways to speed up the MCTS algorithm in the future.

6 CONCLUSION

In this paper, we present a novel and effective approach to learning a vine structure. Our method combines the original MCTS algorithm with the proposed vine UCT, which is adapted from the original UCT for the vine structure learning problem. Under the guidance of the vine UCT, our method can effectively explore the large search space of possible truncated vines by balancing between exploration and exploitation. We demonstrate that the proposed method has significantly better performance on vine structure learning over the existing methods under various experimental setups with Gaussian copula setting. The comparisons have been made on correlation matrices of actual datasets to reflect performance that might be expected in practical applications. Future research can be done to explore the combination of MCTS vine structure learning and non-Gaussian copulas.

References

Bentler, P. M. (1990). Comparative fit indexes in structural models. *Psychological Bulletin*, 107(2):238.

Brechmann, E. C. and Joe, H. (2015). Truncation of vine copulas using fit indices. *Journal of Multivariate Analysis*, 138:19–33.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43.

Chaslot, G. M. J. B., Winands, M. H. M., Herik, H. J. v. D., Uiterwijk, J. W. H. M., and Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357.

Childs, B. E., Brodeur, J. H., and Kocsis, L. (2008). Transpositions and move groups in Monte Carlo tree search. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 389–395. IEEE.

Cooke, R. M., Joe, H., and Chang, B. (2015). Vine regression. *Resources for the Future Discussion Paper*, RFF DP 15-52.

Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer.

Dissmann, J., Brechmann, E. C., Czado, C., and Kurowicka, D. (2013). Selecting and estimating regular vine copulae and application to financial returns. *Computational Statistics & Data Analysis*, 59:52–69.

Elidan, G. (2013). Copulas in machine learning. In *Copulae in Mathematical and Quantitative Finance*, pages 39–60. Springer, Berlin.

Gelly, S. and Wang, Y. (2006). Exploration exploitation in go: UCT for Monte-Carlo go. In *NIPS: Neural Information Processing Systems Conference Online trading of Exploration and Exploitation Workshop*.

Joe, H. (2014). *Dependence Modeling with Copulas*. Chpaman & Hall / CRC Press, Boca Raton, FL.

Kocsis, L. and Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293. Springer.

Kraus, D. and Czado, C. (2017). Growing simplified vine copula trees: improving Dissmann's algorithm. *arXiv preprint arXiv:1703.05203*.

Krupskii, P., Huser, R., and Genton, M. G. (2018). Factor copula models for replicated spatial data. *Journal of the American Statistical Association*, 113(521):467–479.

Kurowicka, D. and Cooke, R. (2003). A parameterization of positive definite matrices in terms of partial correlation vines. *Linear Algebra and its Applications*, 372:225–251.

Kurowicka, D. and Joe, H. (2011). *Dependence Modeling: Vine Copula Handbook*. World Scientific, Singapore.

Ozdemir, O., Allen, T., Choi, S., Wimalajeewa, T., and Varshney, P. (2017). Copula based classifier fusion under statistical dependence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401.

Silver, D., Huang, A., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Sklar, A. (1959). Fonctions de répartition à n dimensions et leurs marges. *Publications de l'Institut de Statistique de l'Université de Paris*, 8:229–231.

Tran, D., Blei, D., and Airoldi, E. M. (2015). Copula variational inference. In *Advances in Neural Information Processing Systems*, pages 3564–3572.

Wieczorek, A., Wieser, M., Murezzan, D., and Roth, V. (2018). Learning sparse latent representations with the deep copula information bottleneck. In *International Conference on Learning Representations*.