

Chapter 1

Counting Vines

O. Morales-Nápoles

*Delft Institute of Applied Mathematics,
Mekelweg 4 Delft, The Netherlands 2628CD
location: HB 06.160*

*O.MoralesNapoles@ewi.tudelft.nl**

In this chapter three algorithms for producing and enumerating regular vines are presented. The first one generates all possible vines on n nodes and regular vines are found by inspection. The second one uses the concept of line graphs to produce only regular vines. The third algorithm produces regular vines by extending a regular vine on three nodes to a regular vine on n nodes. The first and second algorithms presented have been used for the construction of a catalogue of labeled regular vines on at most 9 nodes and tree-equivalent regular vines on at most 7. This catalogue is presented as an appendix to this chapter.

1.1. Introduction.

Regular vines have found application in probability theory and uncertainty analysis. More recently they are becoming popular in statistical analysis of data ([1], [2], [3], [4], [5]). These last references are concerned with choosing an optimal vine to represent multivariate data sets. Algorithms for enumerating all possible $\frac{n!}{2} \times 2^{\binom{n-2}{2}}$ ([6]) regular vines on n nodes will be needed for this purpose.

The problem of counting graphs has been undertaken in the past [7]. Trees are the immediate ancestors of vines and were first successfully counted by Cayley [8]. Trees were used in [9] and [10] as special cases of graphical models, however undirected graphs with cycles were also used.

*Delft Institute of Applied Mathematics. Faculty of Electrical Engineering, Mathematics and Computer Science. Delft University of Technology.

Trees were also used in [11] to infer discrete distributions^a from data.

After introducing some definitions required in the rest of this chapter (section 1.2), previous results about enumeration of trees will be discussed. Algorithms for producing and enumerating regular vines are presented. The first one produces all possible vines on n nodes and regular vines are found by inspection (section 1.3). The second one uses the concept of line graphs to generate only regular vines (section 1.4). The third algorithm generates regular vines by growing a regular vine on three nodes up to a regular vine on n nodes (section 1.5). The first and second algorithms presented have been used for the construction of a catalogue of labeled regular vines on at most 9 nodes and tree-equivalent regular vines on at most 7. This catalogue is presented as an appendix to this chapter. Proofs for the results presented in this chapter have been discussed previously ([6]) and hence are omitted here.

1.2. Basic Definitions.

A *tree* is an undirected acyclic graph. The graph isomorphism problem consists on deciding whether there exists a mapping from the nodes of one graph to the nodes of a second graph such that the edge adjacencies are preserved.

Two *labeled graphs* $G_i = (E_i, N_i)$ and $G_j = (E_j, N_j)$ are *isomorphic* if there is a bijection $\varphi : N_i \rightarrow N_j$ such that for all pairs $(a, b) \in E_i \iff (\varphi(a), \varphi(b)) \in E_j$. If two graphs are isomorphic they are the same *unlabeled* graph.

Graph isomorphism is important in selecting the regular vine that best fits a given data set. For example, the algorithm proposed in [1, p.189] suggests that operations to assign the ‘best’ vine to a data set should begin by selecting the first tree and iteratively for selecting subsequent trees of the regular vine. Once the first tree of a regular vine has been selected, only a fixed number of regular vines may be tested in next steps. Knowing exactly how many regular vines are still possible in next steps and how to construct them might be of advantage for analysis.

The *line graph* $LG(G)$ of a graph G has as its nodes the edges of G , with two nodes being adjacent in LG if the corresponding edges are adjacent in G .

A connected graph $T = (N, E)$ is called a *labeled tree* with nodes $N =$

^aActually the method presented in [11] characterized trees as directed graphs and keeps a close relationship with BBNs (Chapter ??).

$\{1, 2, \dots, n\}$ and edges E , where E is a subset of pairs of N with no cycle. Every sequence of numbers $R(T) = (A_1, A_2, \dots, A_{n-2})$ where each A_i is an integer not greater than n is a *Prüfer Code* for some labeled tree T on n nodes.

A *spanning graph* SS of a graph G is a graph with the same set of nodes as G . If SS is a tree, it is called a *spanning tree* of G .

Trees have been used to represent high dimensional probability distributions [12] and they are often called dependence or markov-dependence trees. For an account of dependence trees see [13]. In dependence trees nodes are associated with random variables with invertible distribution function and arcs are associated with rank correlations realized by bivariate copulae. Figures 1.1 and 1.2 show two different labeled trees with rank correlations attached to their edges. By setting nodes $5 = 2$, $2 = 3$, $4 = 5$ and $3 = 4$ in T_2 it would be transformed into T_1 and hence considered the same unlabeled tree.

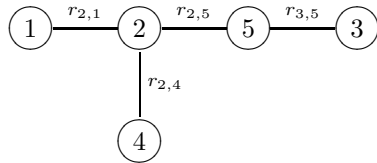


Fig. 1.1. T_1 a tree on 5 nodes with $R(T_1) = (2, 5, 2)$.

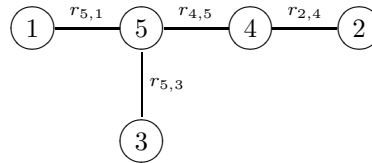


Fig. 1.2. T_2 a tree on 5 nodes with $R(T_2) = (5, 4, 5)$.

A vine [12] is a set of nested trees. Just as labeled trees, vines have been used to represent high dimensional probability distributions [14] and [13] with applications in uncertainty analysis. Vines use sequences of conditional distributions to build a multivariate distribution where conditional bivariate constraints are satisfied. The definitions of *vine* and *regular vine* have been provided in chapter ?? and hence are not repeated here.

As in dependence trees, the nodes of T_1 in a regular vine represent random variables with invertible distribution function. Edges are associated with rank and conditional rank correlations. Figure 1.3 presents the sequence of trees for a regular vine $V_1(5)$ on five nodes. The conditioned set is separated from the conditioning set by a vertical line “|” in the conditional rank correlations from figure 1.3.

Nodes reachable from a given edge in a regular vine are called the *constraint set* of that edge. When two edges are joined by an edge in tree T_i ,

the intersection of the respective constraint sets form the *conditioning set*. The symmetric difference of the constraint sets is the *conditioned set*.

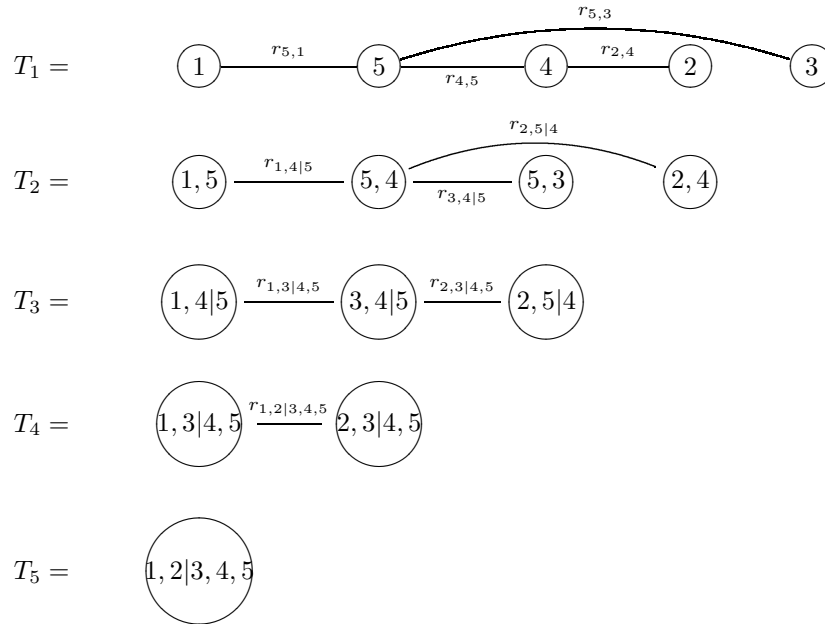


Fig. 1.3. $V_1(5)$ (Regular vine on 5 nodes).

If node e is an element of node f in a regular vine, we say that e is an m -child of f ; similarly, if e is reachable from f via the membership relation: $e \in e_1 \in \dots \in f$, we say that e is an m -descendent of f .

If a bijection may be found for each $T_i \in V_k(n)$ and $T_i \in V_j(n)$ then we speak of the same *tree-equivalent vine* and accordingly the same *tree-equivalent regular vine* when the proximity condition holds. For example setting nodes $5 = 2$, $2 = 3$, $4 = 5$ and $3 = 4$ in T_2 would generate different labeled regular vines but the same tree-equivalent regular vine. Observe that it is easy to find non-regular vines that are tree-equivalent with regular vines.

If element a occurs with element b as conditioned variables in tree k , then a and b are termed k -partners. Nodes A and B are *siblings* if they are m -children of a common parent.

A *natural order* of the elements of a regular vine on n elements is a

Theorem 1.1. *The number of labeled trees on n nodes is n^{n-2} .*

One of various proofs due to [16] of this theorem provides a very useful result for representing labeled trees. The argument is to notice that there is a one to one correspondence between the set of trees with n labeled nodes and the set of Prüfer codes.

In his paper Prüfer obtains the correspondence by the following procedure: For a given tree, remove the endpoint^b with the smallest label (other than the root^c) and let A_1 be the label of the unique node which is adjacent to it. Remove the endpoint and the edge adjacent to it and a tree on $n - 1$ nodes is obtained. Repeat the operation with the new tree on $n - 1$ nodes to obtain A_2 and so on. The process is terminated when a tree on two nodes has been found. The reader may check that the trees from figures 1.1 and 1.2 have Prüfer codes $R(T_1) = (2, 5, 2)$ and $R(T_2) = (5, 5, 2)$ respectively. The procedure described above may be easily reversed, that is, suppose you start with a sequence of $(n - 2)$ -tuples $R(T_k) = (A_1, A_2, \dots, A_{n-2})$ then to obtain the only tree corresponding to the sequence one applies algorithm 1.1:

Algorithm 1.1. Decoding a Prüfer code.

- (1) Take a sequence $R(T_k) = (A_1, A_2, \dots, A_{n-2})$ for $k = 1, 2, \dots, n^{n-2}$ where each A_i , $i = 1, 2, \dots, n - 2$ is an integer not greater than n .
- (2) Write the root in the right most position of $R(T_k)$. Notice that $R(T_k)$ has now length $n - 1$ which is $|E|$.
- (3) Write another row of integers on the bottom of $R(T_k)$ from left to right. Each entry B_i in this new row is the smallest integer that has not been already written in this new row (the row of B_i 's) nor in the first row (the row of A_i 's) in the position exactly above it or every other position to the right.
- (4) The resulting code $S(T_k)$ is the *Extended Prüfer Code*. Each column in the extended Prüfer code represents an arc in the unique labeled tree corresponding to it.

$$S(T_k) = \begin{pmatrix} A_1 & A_2 & A_3 & \dots & n \\ B_1 & B_2 & B_3 & \dots & B_{n-1} \end{pmatrix}$$

^bThe endpoints are nodes with degree one in the tree, they are sometimes referred to as *leafs*.

^cWithout loss of generality we will choose node n as the root of all labeled trees on n nodes. Choosing any other node as the root makes no difference except that the algorithm and the procedure to find the Prüfer code for a given tree must be modified.

Take the two Prüfer codes $R(T_1) = (2, 5, 2)$ and $R(T_2) = (5, 4, 5)$. Apply algorithm 1.1 to decode each sequence into the extended Prüfer code. The reader may check in equation 1.2 that $S(T_1)$ corresponds to figure 1.1 and $S(T_2)$ to figure 1.2.

$$S(T_1) = \begin{pmatrix} 2 & 5 & 2 & 5 \\ 1 & 3 & 4 & 2 \end{pmatrix}, S(T_2) = \begin{pmatrix} 5 & 4 & 5 & 5 \\ 1 & 2 & 3 & 2 \end{pmatrix} \quad (1.2)$$

Prüfer then gives an induction argument to show that for each $(n - 2)$ -tuple there is some tree which determines the given sequence by the above procedure. From the code one can see that a node with degree m would occur exactly $m - 1$ times in the code.

Since every labeled tree can be represented by a Prüfer code, then every tree in a vine may also be represented by a Prüfer code and in this way the vine may be generated. A way to write all possible vines on n nodes is presented in algorithm 1.2.

Algorithm 1.2. Constructing all possible vines on n nodes.

- (1) Set $i = 1$.
- (2) Construct all Prüfer codes possible for T_i .
- (3) The edges of each one of the $n^{n-(i+1)}$ trees in step 2 become nodes in T_{i+1} . Hence, for each tree in step (2):
 - (a) Label the $n - i$ edges of each tree giving the label 1 to the edge appearing in the first column in its extended Prüfer code, 2 to the edge in the second column and so on until all edges have been labeled^d.
 - (b) Construct all Prüfer codes possible for T_{i+1} and connect the new labeled edges (from T_i) as nodes according to these new Prüfer codes.
- (4) Set $i := i + 1$ and go to step (3) until two edges must be connected in the last tree. At this point there is only one way to connect them and no Prüfer code is required.

From algorithm 1.2 it may be observed that to write any vine on n nodes all is required are $n - 2$ Prüfer codes. The first one of length $n - 2$, the

^dThis labeling is not unique and any other labeling would work equally well as long as all n^{n-2} trees are labeled in the same way.

second one of length $n - 3$ and so on until the last one of length 1. A vine on n nodes may be represented by an upper triangular array of size $(n - 2) \times (n - 2)$ whose first row represents the Prüfer code of the first tree in the vine, the second row the second tree of the vine and so on. For example $V_1(5)$ represents the vine from figure 1.3:

$$V_1(5) = \begin{pmatrix} 5 & 4 & 5 \\ & 4 & 4 \\ & & 3 \end{pmatrix} \quad (1.3)$$

Representing a vine with an upper triangular array of size $(n - 2) \times (n - 2)$ provides a convenient way of storing vines. The representation from equation 1.3 provides some idea about the unlabeled tree used at each level in the vine. For example in the first tree node 5 will have degree 3, in the second tree node 4 will also have degree 3 and in the last tree there will be a single node with degree 2 (obviously) which is node 3. A disadvantage is however that the array in equation 1.3 does not tell us right away which node is node 4 in T_2 that from figure 1.3 we may see it is node (5,4). Similarly it is not immediately evident that node 3 in T_3 of equation 1.3 corresponds to (3,4|5) in figure 1.3. Observe that despite a *regular vine array* requires more space for storing, it is more clear regarding the conditioned and conditioning variables used at each level in the vine. Corollary 1.1 follows immediately from the definition of vine and theorem 1.3.

Corollary 1.1. *The number of vines on n nodes is $\prod_{i=1}^n i^{i-2}$.*

Regular vines are most interesting in uncertainty analysis. Implementing Algorithm 1.2 in a computer is very easy and it provides a simple way to construct all possible regular vines on n nodes by simply discarding those that are not regular. However, this method incurs an excessive burden of searching all vines. According to corollary 1.1 the number of vines grows extremely fast with n and it could be very restrictive in time to find all regular vines with algorithm 1.2 even for a modest number of nodes (8 or 9). Another possibility to construct only regular vines will be discussed in the next section.

1.4. Regular Vines and Line Graphs.

The idea is to use the line graph^e of each tree in the vine. Harary notes in [18] that the concept of the line graph of a given graph is so natural that it has been rediscovered independently by many authors.

If the edges of the first tree of figure 1.3 are labeled according to the second step in algorithm 1.2 then the line graph of this tree can be found. This line graph corresponds to figure 1.4. If in the same way we label the edges of the second tree in the vine in figure 1.3 accordingly, then the line graph in figure 1.5 may be obtained.

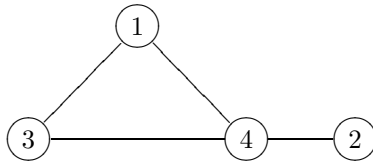


Fig. 1.4. Line Graph of the first tree in figure 1.3

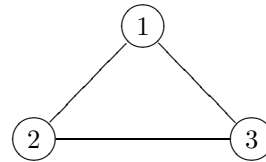


Fig. 1.5. Line Graph of the second tree of the vine from figure 1.3.

It is clear that in order to find all regular vines on n nodes, all the spanning trees of the line graphs of all subtrees in the vine must be found. This result is summarized in algorithm 1.3.

Algorithm 1.3. Constructing all possible regular vines on n nodes based on line graphs.

- (1) Set $i = 1$.
- (2) Construct all Prüfer codes possible for T_i .
- (3) The edges of each one of the $n^{n-(i+1)}$ trees in step (2) become nodes in T_{i+1} . Hence, for each tree in step (2):
 - (a) Label the edges of each tree giving label 1 to the edge appearing in the first column in its extended Prüfer code, 2 to the edge in the second column and so on until all edges have been labeled^f.
- (4) Construct the line graph of each one of the trees from step 2.

^eLine graphs are also known as derived graphs, interchange graphs, adjoint and edge to vertex dual [17].

^fAs before, this labeling is not unique and any other labeling would work equally well as long as all n^{n-i+1} are uniquely labeled.

- (5) For each line graph from step 3 find all possible spanning trees. Connect the edges of each tree in step 1 according to all spanning trees from its line graph. This will give all possible T_{i+1} for each T_i .
- (6) Set $i := i + 1$ and go to step (2) until two edges must be connected in the last tree. At this point there is only one way to connect them and no Prüfer code is required.

Notice that the vines generated by this procedure may still be stored in an $(n - 2) \times (n - 2)$ upper triangular array as in equations 1.3 once a way of labeling the edges from each tree in the vine is specified. Algorithm 1.3 does not produce any irregular vine as opposed to algorithm 1.2. However it involves a greater programming effort and more operations as all possible spanning trees of the line graphs in all trees in the vine must be found. Several algorithms for finding all spanning trees of a given graph have been proposed and examined [19], [20], [21], [22] and [23]. In general finding all possible spanning trees of a given graph other than a complete graph[§] is demanding in terms of time and space [22]. Another algorithm for constructing regular vines without duplication will be presented in next section.

1.5. Regular Vines and Regular Vine Arrays.

One disadvantage of using a triangular array such as the one used in section 1.1 is that the information regarding the label of variables in the first tree of a regular vine is lost when assigning new labels to its edges when they become nodes of the next tree. The same happens as more trees are added to a regular vine. This means that conditioned and conditioning sets are not immediately visible anymore. However a regular vine array preserves the information concerning the labels of the first tree as lower trees in the vine are added.

The *regular vine array* defined in section 1.2 was used in [6] to show that the number of regular vines possible with n nodes is $\frac{n!}{2} \times 2^{\binom{n-2}{2}}$. Example 1.5.1 shows how to construct all possible regular vines on 5 nodes with the natural order $NO(V(5)) = (1, 2, 3, 4, 5)$. This example is useful in showing how to arrive at a general result about the number of labeled regular vines on n nodes.

Example 1.5.1. Constructing regular vines with natural order

[§]For a complete graph all possible spanning trees are the n^{n-2} Prüfer codes

$NO(V(5)) = (1, 2, 3, 4, 5)$.

Observe that the diagonal and off diagonal elements of the *regular vine array* are fixed from the natural order and the definition of *regular vine array*. Element $A_{1,3}$ will also always be fixed by the choices of $A_{3,3}$ and $A_{2,3}$. This means that we start with a regular vine on three nodes. The objective will be to extend this regular vine on three nodes to a regular vine on 5 nodes in all possible ways that preserve regularity within our natural order. Hence, the *regular vine array* with the natural order $NO(V(5)) = (1, 2, 3, 4, 5)$ will look as in equation 1.4 in the beginning.

$$TA(V(5)) = \begin{pmatrix} A_{5,5} \\ A_{4,5} & A_{4,4} \\ A_{3,5} & A_{3,4} & A_{3,3} \\ A_{2,5} & A_{2,4} & A_{2,3} & A_{2,2} \\ A_{1,5} & A_{1,4} & A_{1,3} & A_{1,2} & A_{1,1} \end{pmatrix} = \begin{pmatrix} 1 \\ 2 & 2 \\ A_{3,5} & 3 & 3 \\ A_{2,5} & A_{2,4} & 4 & 4 \\ A_{1,5} & A_{1,4} & 5 & 5 & 5 \end{pmatrix} \quad (1.4)$$

We will start filling in $TA(V(5))$ from top to bottom and from right to left. Hence $A_{2,4}$ will be the next element to be filled in $TA(V(5))$. Observe that element $A_{1,4}$ will be fixed by the choice of $A_{2,4}$. Since we are filling in column number 4 of $TA(V(5))$, from the definition of *regular vine array* $A_{2,4} \in \{A_{3,3}, A_{2,2}, A_{1,1}\}$. However, also from the definition of *regular vine array* $A_{3,3} = A_{3,4}$ and hence $A_{2,4} \in \{A_{3,3}, A_{2,2}, A_{1,1}\} \setminus A_{3,3} = \{A_{2,2}, A_{1,1}\}$. In order to preserve regularity node $A_{3,4} = (3, 2|4, 5)$ must have two children in the lower tree of the vine. These siblings must also have a common child in one tree lower in the vine. By the definition of regular vine array the first child of node $A_{3,4} = (3, 2|4, 5)$ must be node $A_{2,4}$ and its sibling must be in some column $h < 4$ and some row $k \leq 2$, that is in the part of the *regular vine array* that is known. In this case nodes $A_{2,4}$ and $A_{2,3} = (3, 4|5)$ must be siblings and must have common child $A_{1,2} = (5, 4)$. Observe that if element $A_{2,4} = 4$ then nodes $A_{2,4} = (4, 2|5)$ and $A_{2,3} = (3, 4|5)$ will be siblings and have common child $A_{1,2} = (5, 4)$. If element $A_{2,4} = 5$ then nodes $A_{2,4} = (5, 2|4)$ and $A_{2,3} = (3, 4|5)$ will be siblings and have common child $A_{1,2} = (5, 4)$. Hence either $A_{2,4} = A_{2,2}$ or $A_{2,4} = A_{1,1}$ preserves regularity and fixes immediately element $A_{1,4}$. Then both *regular vine arrays* in equation 1.5 are possible.

$$TA_a(V(5)) = \begin{pmatrix} 1 \\ 2 & 2 \\ A_{3,5} & 3 & 3 \\ A_{2,5} & 5 & 4 & 4 \\ A_{1,5} & 4 & 5 & 5 & 5 \end{pmatrix} \quad TA_b(V(5)) = \begin{pmatrix} 1 \\ 2 & 2 \\ A_{3,5} & 3 & 3 \\ A_{2,5} & 4 & 4 & 4 \\ A_{1,5} & 5 & 5 & 5 & 5 \end{pmatrix} \quad (1.5)$$

Next element $A_{3,5}$ must be found for both $TA_a(V(5))$ and $TA_b(V(5))$ in equation 1.5. From a similar argument as before, $A_{3,5} \in \{A_{3,3}, A_{2,2}, A_{1,1}\}$ for $TA_a(V(5))$ and $TA_b(V(5))$. Consider first $TA_a(V(5))$. Node $A_{3,5}$ and $A_{3,4} = (3, 2|4, 5)$ have common parent $A_{2,5} = (2, 1|3, 4, 5)$ and hence are siblings. Nodes $A_{3,5}$ and $A_{3,4}$ must have a common child in the lower tree in order to keep regularity. Possible candidates are nodes $A_{2,4} = (5, 2|4)$ or $A_{2,3} = (4, 3|5)$. However, element $A_{4,4}$ is not an element of node $A_{3,5}$ hence $A_{2,3} = (4, 3|5)$ must be the sibling of node $A_{2,5}$. And they must have a common child in the lower tree. The tree possible choices for $A_{3,5}$ are listed next.

- (1) $A_{3,5} = A_{3,3} = 3$, then either $A_{2,5} = (1, 4|5)$ or $A_{2,5} = (1, 5|4)$. $A_{2,5} = (4, 5|1)$ is not a valid choice because element $A_{5,5} = 1$ and it must be in the conditioned set of every node in column 5. Nodes $A_{2,3} = (4, 3|5)$ and $A_{2,5}$ must have a common child in the lower tree. This must be either $A_{1,3} = (5, 3)$ or $A_{1,2} = (5, 4)$. Element $A_{3,3} = 3$ is not an element of node $A_{2,5}$ from the definition of *regular vine array* and the assumption that $A_{3,5} = A_{3,3} = 3$. Hence node $A_{1,2} = (5, 4)$ must be the common child. If node $A_{2,5} = (1, 4|5)$ then nodes $A_{1,5} = (5, 1)$ and $A_{1,2} = (5, 4)$ are its children and regularity is preserved. On the other hand if node $A_{2,5} = (1, 5|4)$ then nodes $A_{1,5} = (4, 1)$ and $A_{1,2} = (5, 4)$ are its children and regularity is again preserved. Hence element $A_{3,3} = 3$ is a valid choice for $A_{3,5}$ and both matrices in equation 1.6 are possible. Observe also that given element $A_{3,5} = 3$ there are two possible choices for element $A_{2,5}$. That is either $A_{2,5} = 4$ or $A_{2,5} = 5$ and either choice maintains regularity. $A_{1,5}$ is fixed by our previous choices.

$$TA_1(V(5)) = \begin{pmatrix} 1 \\ 2 & 2 \\ 3 & 3 & 3 \\ 4 & 5 & 4 & 4 \\ 5 & 4 & 5 & 5 & 5 \end{pmatrix} \quad TA_2(V(5)) = \begin{pmatrix} 1 \\ 2 & 2 \\ 3 & 3 & 3 \\ 5 & 5 & 4 & 4 \\ 4 & 4 & 5 & 5 & 5 \end{pmatrix} \quad (1.6)$$

- (2) $A_{3,5} = A_{2,2} = 4$, then either $A_{2,5} = (1, 5|3)$ or $A_{2,5} = (1, 3|5)$. $A_{2,5} = (3, 5|1)$ is not a valid choice because element $A_{5,5} = 1$ and it must be in the conditioning set of every node in column 5. Nodes $A_{2,3} = (4, 3|5)$ and $A_{2,5}$ must have a common child in the lower tree. This must be either $A_{1,3} = (5, 3)$ or $A_{1,2} = (5, 4)$. Element $A_{2,2} = 4$ is not an element of node $A_{2,5}$ from the definition of *regular vine array* and the assumption that $A_{3,5} = A_{3,3} = 4$. Hence node $A_{1,3} = (5, 3)$ must be the common child. If node $A_{2,5} = (1, 5|3)$ then nodes $A_{1,5} = (3, 1)$ and $A_{1,3} = (5, 3)$ are its children and regularity is preserved. On the other hand if node $A_{2,5} = (1, 3|5)$ then nodes $A_{1,5} = (5, 1)$ and $A_{1,3} = (5, 3)$ are its children and regularity is again preserved. Hence element $A_{2,2} = 4$ is a valid choice for $A_{3,5}$ and both matrices in equation 1.7 are possible. Observe also that given element $A_{3,5} = 4$ there are two possible choices for element $A_{2,5}$. That is either $A_{2,5} = 3$ or $A_{2,5} = 5$ and either choice maintains regularity. $A_{1,5}$ is fixed by our previous choices.

$$TA_3(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 4 \ 3 \ 3 \\ 5 \ 5 \ 4 \ 4 \\ 3 \ 4 \ 5 \ 5 \ 5 \end{pmatrix} TA_4(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 4 \ 3 \ 3 \\ 3 \ 5 \ 4 \ 4 \\ 5 \ 4 \ 5 \ 5 \ 5 \end{pmatrix} \quad (1.7)$$

- (3) $A_{3,5} = A_{1,1} = 5$, then there are two possibilities, either $A_{2,5} = (3, 1|4)$ or $A_{2,5} = (4, 1|3)$. Nodes $A_{2,3} = (4, 3|5)$ and $A_{2,5}$ must have a common child in the lower tree. This must be either $A_{1,3} = (5, 3)$ or $A_{1,2} = (5, 4)$ however element $A_{1,3} = A_{1,2} = 5$ is not an element of node $A_{2,5}$ from the definition of *regular vine array* and the assumption that $A_{3,5} = A_{1,1} = 5$. Hence element $A_{1,1} = 5$ is not a valid choice for $A_{3,5}$.

By a similar procedure as described earlier the reader may check that $TA_b(V(5))$ in equation 1.5 may be extended to the four regular vines in equations 1.8 and 1.9.

$$TA_5(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 3 \ 3 \ 3 \\ 4 \ 4 \ 4 \ 4 \\ 5 \ 5 \ 5 \ 5 \ 5 \end{pmatrix} TA_6(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 3 \ 3 \ 3 \\ 5 \ 4 \ 4 \ 4 \\ 4 \ 5 \ 5 \ 5 \ 5 \end{pmatrix} \quad (1.8)$$

$$TA_7(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 4 \ 3 \ 3 \\ 5 \ 4 \ 4 \ 4 \\ 3 \ 5 \ 5 \ 5 \ 5 \end{pmatrix} TA_8(V(5)) = \begin{pmatrix} 1 \\ 2 \ 2 \\ 4 \ 3 \ 3 \\ 3 \ 4 \ 4 \ 4 \\ 5 \ 5 \ 5 \ 5 \ 5 \end{pmatrix} \quad (1.9)$$

Summarizing, we may see that for every one of the two choices for $A_{2,4}$ that keep regularity, there are two choices for $A_{3,5}$ that will keep regularity. Again, for each of the two choices of $A_{3,5}$ there will also be two possible choices for $A_{2,5}$ that will keep regularity. In other words there are $2 \times 2 \times 2 = 2^3 = 8$ regular vines possible with the natural order $NO(V(5)) = (1, 2, 3, 4, 5)$. \square

In [6] an argument similar to the one presented in example 1.5.1 (using induction on n) is used to show that the number of regular vines possible with a fixed natural order $NO(V(n)) = A_{n,n}, A_{n-1,n-1}, \dots, A_{1,1}$ is: $\prod_{j=1}^{n-3} 2^j = 2^{\binom{n-2}{2}}$. It is easy to see that there are $\binom{n}{2}$ ways of choosing the pair $A_{n,n}, A_{n-1,n-1}$ in a natural order and $(n-2)!$ ways of permuting elements $A_{n-2,n-2}, \dots, A_{1,1}$. Hence there are $\frac{n!}{2} \times 2^{\binom{n-2}{2}}$ labeled regular vines on n nodes.

The procedure to find regular vines from *regular vine arrays* presented in example 1.5.1 also provides another algorithm for constructing regular vines without duplication. The algorithm is presented next. The basic idea is first to construct all possible natural orders on n nodes and then use them to build all *regular vines arrays* possible.

Algorithm 1.4. Constructing all possible regular vine arrays on n nodes.

- (1) For $n \leq 3$ constructing regular vines is trivial hence consider $n \geq 4$.
- (2) Create the $\binom{n}{2} \times (n-2)! = \frac{n!}{2}$ natural orders possible on n nodes.
- (3) For each of the natural orders in step (1):
 - (a) Generate the regular vine on three nodes that corresponds to the natural order as in equation 1.1.
 - (b) Set $c := 4$ $r := 2$
 - (c) Find the 2 possibilities of selecting each one of $A_{c-2,r}, \dots, A_{2,c}$ that would preserve regularity given the previous choices. Each choice should be in $\{A_{c-2,c-2}, \dots, A_{1,1}\}$ in the natural order.
 - (d) If $c = n$ stop else, set $c := c + 1$ and $r := r + 1$ go to (b)

- (e) The $\prod_{j=1}^{n-3} 2^j = 2^{\binom{n-2}{2}}$ possibilities of building a regular vine array given the natural order have been found.

Algorithm 1.4 does not require the additional operations that algorithm 1.3 require for building line graphs and finding spanning trees for each tree in each level of the regular vine. However it still requires a search in $A_{c-2,c-2}, \dots, A_{1,1}$ in the natural order for selecting choices for each $A_{c-2,c}, \dots, A_{2,c}$ in the regular vine array. Additionally, the construction of the regular vine array provides a natural way of enumerating regular vines. Next, the classification of regular vines is discussed.

1.6. Classifying Regular Vines.

Organizing regular vines in a systematic way may be of advantage. One natural way to start classifying regular vines is according to their equivalence class as shown in chapter ?? by Joe. Another natural way to classify them is according to the unlabeled trees used in their construction. Appendix A is presented as a first step towards a better organization of regular vines. This appendix presents a catalogue of labeled regular vines on at most 9 nodes organized according to the unlabeled tree used in the first tree of the regular vine. It also presents tree-equivalent regular vines on at most 7 nodes. In principle any one of the three algorithms for generating regular vines presented in this chapter may be used to classify regular vines. Algorithms 1.2 and 1.3 were used for the construction of the catalogue presented here.

The names of trees from tables A.1 and A.2 used in each level of each regular vine in tables A.8 to A.11 will be displayed in order after the + sign. There is one tree-equivalent regular vine on 3 nodes: $V3 = T3 + T2 + T1$. Every regular vine on n nodes for $n > 3$ must necessarily use $V3$ in its construction. For this reason $T3 + T2 + T1$ will be omitted when indicating the sequence of trees used in the construction of different tree-equivalent regular vines. For example the D-vine on 4 nodes will be $V4 = T4 + V3 = T4$.

The 8 regular vines generated in example 1.5.1 may be classified according to their equivalence and tree-equivalent classes. According to table A.8 in appendix A there are 5 possible tree sequences for regular vines on 5 nodes. These 5 sequences are shown in the first column of table 1.1. The second column apportions the regular vines from example 1.5.1 to these 5

tree sequences. The names of trees from table A.1 used in each level of each regular vine in example 1.5.1 are displayed as explained before.

Table 1.1. Apportioning regular vines from example 1.5.1 to tree-equivalent classes.

| Tree sequence | Vines from example 1.5.1 within given tree sequence |
|---------------|---|
| T6+T4 | TA_3 |
| T7+T4 | TA_4, TA_7 |
| T7+T5 | TA_1, TA_2, TA_6 |
| T8+T4 | TA_8 |
| T8+T5 | TA_5 |

The reader may check that TA_3 corresponds to a D-Vine and TA_5 to a C-Vine. TA_8 has one node with maximal degree in its first tree and a D-Vine on four nodes is attached to this first tree. It is evident that these three vines besides corresponding to different tree-equivalent classes correspond to different equivalent classes.

TA_4 and TA_7 besides being tree-equivalent they are in the same equivalence class. The reader may check this by permuting nodes $4 \leftrightarrow 3$ and $2 \leftrightarrow 1$ in either TA_4 or TA_7 to obtain the same vine.

TA_1, TA_2 and TA_6 are tree equivalent. By making node $2 = 1, 5 = 4, 3 = 2$ and $1 = 3$ in TA_6 it becomes TA_2 and hence these two are in the same equivalence class. However, TA_1 cannot be transformed into either TA_2 or TA_6 by a permutation of nodes in the first tree. Hence, TA_1 falls in a different equivalence class than TA_2 and TA_6 despite the fact that the three are tree-equivalent. Observe that the 6 equivalence classes for regular vines on 5 nodes mentioned in chapter?? by Joe are represented in example 1.5.1.

According to results from previous section there are $\frac{5!}{2} = 60$ other possible natural orders than the one used in example 1.5.1. Hence there must be 60×1 D-Vines and C-Vines. Also, 60×2 regular vines in the class of TA_4 and TA_7 must be observed. Finally there must be 60×3 regular vines with tree-sequence T7+T5. This result may also be observed in V6-V10 in table A.8 in appendix A. Observe that after the classification of the *regular vine arrays* from example 1.5.1, from the 180 regular vines with tree-sequence T7+T5, 60 must be in the same equivalence class as TA_1 . Appendix A was elaborated without the implementation of *regular vine arrays*. This verifies that it is possible to arrive at the same conclusions with the three different methods proposed in present chapter. Finally, observe that it is sufficient

to investigate one natural order to classify regular vines within equivalence and tree-equivalent classes.

1.7. Conclusions and Final Comments.

A way to efficiently code and store vines on n nodes based on the Prüfer code is proposed. This consists of an upper triangular matrix of size $(n - 2) \times (n - 2)$. An algorithm for building vines and two others for building regular vines on n nodes have been presented. Algorithm 1.2 is easy to implement and efficient if regular vines on less than 6 nodes are required. Algorithms 1.3 and 1.4 would produce only regular vines at the cost of greater programming effort and a larger number of arithmetic operations. Tables A.1 to A.7 present the number of labeled trees, regular vines per labeled tree and tree-equivalent regular vines according to unlabeled trees on at most 9 nodes. Tables A.1 and A.2 present the 25 trees on 7 nodes or less. These trees will be used to present pictures of tree-equivalent regular vines on at most 6 nodes in tables A.8 and A.9. Finally tables A.10 and A.11 present tree-equivalent regular vines on 7 nodes.

We have made a first step towards organizing vines and regular vines in a more systematic way. We believe that this task is necessary in order to progress more rapidly the space of applications for vines and make them more accessible for people interested in the subject. Hence our recommendation is to enhance efforts for a more systematic organization of vines including algorithms for generating and storing them.

References

- [1] A. K., C. C., F. A., and B. H, Pair-copula constructions of multiple dependence, *To appear in Insurance: Mathematics and Economics*. **44**(1), (2007).
- [2] A. K. and B. D., Models for construction of multivariate dependence, *Accepted for publication in European Journal of Finance*. (2009).
- [3] A. Min and C. C., Bayesian inference for multivariate copulas using pair copula constructions, *Submitted for publication*. (2008).
- [4] K. O. and S. M. The d-vine creation of non-gaussian random fields. In *GEOSTATS*, (2008).
- [5] L. Chollete, A. Heinen, and A. Valdesogo. Modeling international financial returns with a multivariate regime switching copula. CORE Discussion Papers 2008013, Universit catholique de Louvain, Center for Operations Research and Econometrics (CORE) (Mar., 2008). URL <http://ideas.repec.org/p/cor/louvco/2008013.html>.
- [6] O. Morales-Nápoles, R. Cooke, and D. Kurowicka, About the number of

- vines and regular vines on n nodes, *Submitted to Linear Algebra and its Applications*.
- [7] H. F. and P. E.M., *Graphical Enumeration*. (Academic Press, 1973).
 - [8] A. Cayley, A theorem on trees, *The Quarterly Journal of Pure and Applied Mathematics*. **23**, 376–378, (1889).
 - [9] J. N. Darroch, S. L. Lauritzen, and T. P. Speed, Markov fields and log-linear interaction models for contingency tables, *The Annals of Statistics*. **8** (3), 522–539, (1980). ISSN 00905364. URL <http://www.jstor.org/stable/2240590>.
 - [10] T. P. Speed and H. T. Kiiveri, Gaussian markov distributions over finite graphs, *The Annals of Statistics*. **14**(1), 138–150, (1986). ISSN 00905364. URL <http://www.jstor.org/stable/2241271>.
 - [11] C. Chow and C. Liu, Approximating discrete probability distributions with dependence trees, *Information Theory, IEEE Transactions on*. **14**(3), 462–467, (1968).
 - [12] C. R.M. Markov and entropy properties of tree and vine-dependent variables. In *Proceedings of the ASA Section on Bayesian Statistical Science*, (1997).
 - [13] K. D. and C. R.M., *Uncertainty Analysis with High Dimensional Dependence Modelling*. (Wiley, 2006).
 - [14] B. T.J. and C. R.M., Vines - a new graphical model for dependent random variables, *Ann. of Stat.* **30**(4), 1031–1068, (2002).
 - [15] J. Moon. Various proofs of cayley's formula for counting trees. In ed. F. Harary, *A Seminar on Graph Theory*.
 - [16] V. H. Prüfer, Neuer beweis eines satzes über permutationen, *Arch. Math. Phys.* (27), 742–744, (1918).
 - [17] L. Beineke. Derived graphs with derived complements. In *Recent Trends in Graph Theory: Proceedings of the First New York City Graph Theory Conference held on June 11, 12, and 13, 1970*. Springer, (2006).
 - [18] F. Harary, *Graph Theory*. (Addison-Wesley, 1969).
 - [19] G.J.Minty, A simple algorithm for listing all the trees of a graph, *IEEE Transactions on Circuit Theory*. **12**, 120–120, (1965).
 - [20] W. Mayeda and S. Seshu, Generation of trees without duplication, *IEEE Transactions on Circuit Theory*. **12**, 181–185, (1967).
 - [21] R. Read and R.E.Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks*. **5**, 678–692, (1975).
 - [22] M. J. Smith. Generating spanning trees. Master's thesis, University of Victoria, (1997).
 - [23] A. T. A. Shioura and T. Uno, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks*. **5**, (1975).
 - [24] K. V.N. and E. V.A., *Graph Theory for Programmers-Algorithms for Processing Trees*. (Kluwer Academic Publishers, 2000).

Appendix A. A Catalogue of Labeled Regular Vines on at most 9 nodes & Tree Equivalent Regular Vines on at most 7 nodes.

Lets go now from the zoo of reality to the zoo of mythologies, the garden whose fauna is not of lions but of sphinxes, griffins and centaurs. The population of the second garden should exceed that of the first; since a monster is no other thing than a combination of elements of real beings and the possibilities of the combinatorial art border with the infinite.

Manual de zoología fantástica
J.L. BORGES

The purpose of this catalogue is to classify regular vines according to their graphical structure. We hope that this catalogue will help researchers interested in regular vines with their investigations. Like the authors of [?] this author has “tried that the data is free of errors, but accept[s] no responsibility for any loss of time, money, patience or temper occurring as a result of any mistakes that may have crept into the pages of this [catalogue]. Furthermore, [the author] wishes it to be understood that any mistakes are entirely the fault of the other author.”

Tables A.1-A.7 present the 95 trees on 7 nodes or less. Catalogues of trees on at most twelve nodes have been presented before. In [15] pictures for trees with at most five nodes are presented. In [24] a catalogue of trees with at most 8 nodes may be found^h. [18] presents trees on at most 10 nodesⁱ. The 987 trees on at most 12 vertices (together with about 10,000 other graphs and many tables of interest for graph theorists) may be found in [?]. None of the above catalogues presents results concerning vines.

Vines will be presented by pictures in next section and the names of the trees from table A.1 and A.2 used in each level of each regular vine in tables A.8 to A.11 will be displayed in order after the + sign. There is one tree-equivalent regular vine on 3 nodes $V3 = T3 + T2 + T1$. Every regular vine on n nodes for $n > 3$ must necessarily use $V3$ in its construction. For this reason $T3 + T2 + T1$ will be omitted when indicating the sequence

^hThis catalogue repeats a tree in eight nodes neglecting another one. In the same reference tables with the number of non-isomorphic trees on less than 26 nodes may be found.

ⁱHarary refers to [?] for diagrams of trees with at most 12 nodes. However this reference is not available to the author at the moment of the publication of this catalogue.

of trees used in the construction of different tree-equivalent regular vines. For example the D-vine on 4 nodes will be $V_4 = T_4 + V_3 = T_4$. Next the catalogue is presented.

Counting Vines

Table A.1. Trees with at most 7 nodes.



















| | | | | | | |
|-------------------------------------|---|---|---|---|---|---|
| Prüfer code example | | | 1 | 12 | 11 | 123 |
| |  |  |  |  |  |  |
| | T1 | T2 | T3 | T4 | T5 | T6 |
| # Labeled Trees | 1 | 1 | 3 | 12 | 4 | 60 |
| # Regular Vines per labeled tree | 1 | 1 | 1 | 1 | 3 | 1 |
| # Tree-Equivalent Reg. Vines / tree | 1 | 1 | 1 | 1 | 1 | 1 |
| Prüfer code example | 112 | 111 | 1234 | 1123 | 1213 | 2244 |
| |  |  |  |  |  |  |
| | T7 | T8 | T9 | T10 | T11 | T12 |
| # Labeled Trees | 60 | 5 | 360 | 360 | 360 | 90 |
| # Regular Vines per labeled tree | 5 | 24 | 1 | 7 | 11 | 48 |
| # Tree-Equivalent Reg. Vines / tree | 2 | 2 | 1 | 3 | 3 | 5 |
| Prüfer code example | 1112 | 1111 | 12345 | 12344 | 12234 | 12324 |
| |  |  |  |  |  |  |
| | T13 | T14 | T15 | T16 | T17 | T18 |
| # Labeled Trees | 120 | 6 | 2,520 | 2,520 | 5,040 | 840 |
| # Regular Vines per labeled tree | 75 | 480 | 1 | 9 | 19 | 33 |
| # Tree-Equivalent Reg. Vines / tree | 5 | 5 | 1 | 4 | 7 | 3 |

Table A.2. Trees with at most 7 nodes (Continuation).








| | | | | |
|-------------------------------------|---|---|--|---|
| Prüfer code example | 11233 | 11223 | 11123 | 12223 |
| |  |  |  |  |
| | T19 | T20 | T21 | T22 |
| # Labeled Trees | 630 | 2,520 | 840 | 1,260 |
| # Regular Vines per labeled tree | 80 | 168 | 168 | 342 |
| # Tree-Equivalent Reg. Vines / tree | 9 | 17 | 12 | 17 |
| Prüfer code example | 11122 | 11112 | 11111 | |
| |  |  |  | |
| | T23 | T24 | T25 | |
| # Labeled Trees | 420 | 210 | 7 | |
| # Regular Vines per labeled tree | 1,452 | 2,928 | 23,040 | |
| # Tree-Equivalent Reg. Vines / tree | 22 | 22 | 22 | |

Table A.3. Trees with at most 8 nodes.













| | | | | | | |
|-------------------------------------|--|--|--|--|--|--|
| Prüfer code example | 123456  T26 | 123455  T27 | 122345  T28 | 123345  T29 | 123435  T30 | 112324  T31 |
| # Labeled Trees | 20,160 | 20,160 | 40,320 | 20,160 | 20,160 | 10,080 |
| # Regular Vines per labeled tree | 1 | 11 | 29 | 39 | 71 | 820 |
| # Tree-Equivalent Reg. Vines / tree | 1 | 5 | 12 | 8 | 10 | 44 |
| Prüfer code example | 112344  T32 | 122344  T33 | 122334  T34 | 123344  T35 | 112233  T36 | 122324  T37 |
| # Labeled Trees | 5,040 | 20,160 | 20,160 | 20,160 | 5,040 | 6,720 |
| # Regular Vines per labeled tree | 120 | 315 | 815 | 423 | 4,520 | 2,181 |
| # Tree-Equivalent Reg. Vines / tree | 14 | 38 | 55 | 41 | 72 | 44 |

Table A.4. Trees with at most 8 nodes (Continuation).












| | | | | | | |
|-------------------------------------|---|---|---|---|---|---|
| Prüfer code example | 244466 | 123444 | 123334 | 112333 | 122333 | 111222 |
| |  |  |  |  |  |  |
| | T38 | T39 | T40 | T41 | T42 | T43 |
| # Labeled Trees | 10,080 | 6,720 | 20,160 | 3,360 | 6,720 | 560 |
| # Regular Vines per labeled tree | 11,246 | 315 | 1,046 | 3,384 | 8,667 | 89,712 |
| # Tree-Equivalent Reg. Vines / tree | 114 | 24 | 61 | 72 | 111 | 133 |
| Prüfer code example | 122223 | 123333 | 112222 | 122222 | 222222 | |
| |  |  |  |  |  | |
| | T44 | T45 | T46 | T47 | T48 | |
| # Labeled Trees | 3,360 | 1,680 | 840 | 336 | 8 | |
| # Regular Vines per labeled tree | 27,222 | 11,160 | 117,072 | 279,000 | 2,580,480 | |
| # Tree-Equivalent Reg. Vines / tree | 114 | 83 | 136 | 136 | 136 | |

Table A.5. Trees with 9 nodes.



















| | | | | | | |
|-------------------------------------|---|---|---|--|---|---|
| Prüfer code example | 2345678 | 2345578 | 2345668 | 2345677 | 2345658 | 2345478 |
| |  |  |  |  |  |  |
| | T49 | T50 | T51 | T52 | T53 | T54 |
| # Labeled Trees | 181,440 | 362,880 | 362,880 | 181,440 | 181,440 | 181,440 |
| # Regular Vines on each tree | 1 | 69 | 41 | 13 | 129 | 181 |
| # Tree-Equivalent Reg. Vines / tree | 1 | 21 | 18 | 6 | 22 | 18 |
| Prüfer code example | 2345477 | 2335658 | 2343677 | 2335668 | 2344668 | 2245677 |
| |  |  |  |  |  |  |
| | T55 | T56 | T57 | T58 | T59 | T60 |
| # Labeled Trees | 181,440 | 181,440 | 90,720 | 181,440 | 362,880 | 45,360 |
| # Regular Vines on each tree | 2,651 | 5,390 | 1,708 | 1,646 | 2,708 | 168 |
| # Tree-Equivalent Reg. Vines / tree | 164 | 203 | 104 | 125 | 221 | 20 |
| Prüfer code example | 2335677 | 2344677 | 2345577 | 2344478 | 2345558 | 2345666 |
| |  |  |  |  |  |  |
| | T61 | T62 | T63 | T64 | T65 | T66 |
| # Labeled Trees | 181,440 | 181,440 | 181,440 | 90,720 | 181,440 | 60,480 |
| # Regular Vines on each tree | 528 | 887 | 887 | 4,202 | 2,567 | 528 |
| # Tree-Equivalent Reg. Vines / tree | 70 | 105 | 91 | 147 | 162 | 42 |

Table A.6. Trees with 9 nodes (Continuation).



















| | | | | | | |
|-------------------------------------|---|---|---|---|---|---|
| Prüfer code example | 2345448 | 2343638 | 2245577 | 2335577 | 2245477 | 2344438 |
| |  |  |  |  |  |  |
| | T67 | T68 | T69 | T70 | T71 | T72 |
| # Labeled Trees | 181,440 | 15,120 | 90,720 | 181,440 | 45,360 | 30,240 |
| # Regular Vines on each tree | 8,738 | 18,504 | 11,296 | 34,417 | 36,892 | 72,546 |
| # Tree-Equivalent Reg. Vines / tree | 275 | 99 | 287 | 628 | 350 | 428 |
| Prüfer code example | 2343377 | 2225668 | 2333668 | 2344666 | 2225677 | 2333677 |
| |  |  |  |  |  |  |
| | T73 | T74 | T75 | T76 | T77 | T78 |
| # Labeled Trees | 90,720 | 60,480 | 181,440 | 60,480 | 30,240 | 90,720 |
| # Regular Vines on each tree | 120,444 | 20,904 | 99,028 | 34,143 | 6,756 | 32,812 |
| # Tree-Equivalent Reg. Vines / tree | 724 | 332 | 840 | 439 | 166 | 516 |
| Prüfer code example | 2344477 | 2345555 | 2344448 | 2333637 | 2244666 | 2244477 |
| |  |  |  |  |  |  |
| | T79 | T80 | T81 | T82 | T83 | T84 |
| # Labeled Trees | 90,720 | 15,120 | 60,480 | 30,240 | 30,240 | 22,680 |
| # Regular Vines on each tree | 54,004 | 32,688 | 149,901 | 360,084 | 428,388 | 680,576 |
| # Tree-Equivalent Reg. Vines / tree | 607 | 245 | 765 | 724 | 980 | 1,034 |

Table A.7. Trees with 9 nodes (Continuation).












| | | | | |
|-------------------------------------|---|---|---|--|
| Prüfer code example | 2225666  T85 | 2333666  T86 | 2245555  T87 | 2333377  T88 |
| # Labeled Trees | 5,040 | 30,240 | 7,560 | 30,240 |
| # Regular Vines on each tree | 262,080 | 1,232,820 | 414,432 | 1,919,610 |
| # Tree-Equivalent Reg. Vines / tree | 465 | 1,328 | 735 | 1,328 |
| Prüfer code example | 2335555  T89 | 2344444  T90 | 2333338  T91 | 2225555  T92 |
| # Labeled Trees | 15,120 | 3,024 | 7,560 | 2,520 |
| # Regular Vines on each tree | 1,232,340 | 1,869,120 | 5,255,904 | 14,889,744 |
| # Tree-Equivalent Reg. Vines / tree | 1,195 | 901 | 1,328 | 1,464 |
| Prüfer code example | 2244444  T93 | 2333333  T94 | 1111111  T95 | |
| # Labeled Trees | 1,512 | 504 | 9 | |
| # Regular Vines on each tree | 23,334,480 | 62,523,360 | 660,602,880 | |
| # Tree-Equivalent Reg. Vines / tree | 1,464 | 1,464 | 1,464 | |

Table A.8. Tree-equivalent regular vines with at most 6 nodes.


















| | | | | |
|---|---|--|---|---|
|  |  |  |  |  |
| $V_1 = T_1$ 1 | $V_2 = T_2$ 1 | $V_3 = T_3 + T_2 + T_1$ 3 | $V_4 = T_4$ 12 | $V_5 = T_5$ 12 |
|  |  |  | | |
| $V_6 = T_6 + T_4$ 60 | $V_7 = T_7 + T_4$ 120 | $V_8 = T_7 + T_5$ 180 | | |
|  |  |  | | |
| $V_9 = T_8 + T_4$ 60 | $V_{10} = T_8 + T_5$ 60 | $V_{11} = T_9 + T_6 + T_4$ 360 | | |
|  |  |  | | |
| $V_{12} = T_{10} + T_6 + T_4$ 720 | $V_{13} = T_{10} + T_7 + T_4$ 720 | $V_{14} = T_{10} + T_7 + T_5$ 1,080 | | |
|  |  |  | | |
| $V_{15} = T_{11} + T_6 + T_4$ 360 | $V_{16} = T_{11} + T_7 + T_4$ 1,440 | $V_{17} = T_{11} + T_7 + T_5$ 2,160 | | |

Table A.9. Tree-equivalent regular vines with at most 6 nodes (Continuation).


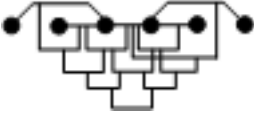













| | | |
|---|---|--|
|  |  |  |
| $V18 = T12+T6+T4$ 360 | $V19 = T12+T7+T4$ 720 | $V20 = T12+T7+T5$ 1,080 |
|  |  |  |
| $V21 = T12+T8+T4$ 1,080 | $V22 = T12+T8+T5$ 1,080 | $V23 = T13+T6+T4$ 720 |
|  |  |  |
| $V24 = T13+T7+T4$ 2,160 | $V25 = T13+T7+T5$ 3,240 | $V26 = T13+T8+T4$ 1,440 |
|  |  |  |
| $V27 = T13+T8+T5$ 1,440 | $V28 = T14+T6+T4$ 360 | $V29 = T14+T7+T4$ 720 |
|  |  |  |
| $V30 = T14+T7+T5$ 1,080 | $V31 = T14+T8+T4$ 360 | $V32 = T14+T8+T5$ 360 |

Table A.10. Tree-equivalent regular vines with 7 nodes.

| Tree sequence & # Tree-equivalent Labeled Regular Vines | | Tree sequence & # Tree-equivalent Labeled Regular Vines | |
|--|--------|--|--------|
| V33 = T15+T9+T6+T4 | 2,520 | V68 = T20+T12+T8+T5 | 30,240 |
| V34 = T16+T9+T6+T4 | 5,040 | V69 = T20+T13+T6+T4 | 15,120 |
| V35 = T16+T10+T6+T4 | 5,040 | V70 = T20+T13+T7+T4 | 45,360 |
| V36 = T16+T10+T7+T4 | 5,040 | V71 = T20+T13+T7+T5 | 68,040 |
| V37 = T16+T10+T7+T5 | 7,560 | V72 = T20+T13+T8+T4 | 30,240 |
| V38 = T17+T9+T6+T4 | 5,040 | V73 = T20+T13+T8+T5 | 30,240 |
| V39 = T17+T10+T6+T4 | 10,080 | V74 = T21+T9+T6+T4 | 5,040 |
| V40 = T17+T10+T7+T4 | 10,080 | V75 = T21+T10+T6+T4 | 5,040 |
| V41 = T17+T10+T7+T5 | 15,120 | V76 = T21+T10+T7+T4 | 5,040 |
| V42 = T17+T11+T6+T4 | 5,040 | V77 = T21+T10+T7+T5 | 7,560 |
| V43 = T17+T11+T7+T4 | 20,160 | V78 = T21+T11+T6+T4 | 5,040 |
| V44 = T17+T11+T7+T5 | 30,240 | V79 = T21+T11+T7+T4 | 20,160 |
| V45 = T18+T11+T6+T4 | 2,520 | V80 = T21+T11+T7+T5 | 30,240 |
| V46 = T18+T11+T7+T4 | 10,080 | V81 = T21+T13+T6+T4 | 5,040 |
| V47 = T18+T11+T7+T5 | 15,120 | V82 = T21+T13+T7+T4 | 15,120 |
| V48 = T19+T9+T6+T4 | 2,520 | V83 = T21+T13+T7+T5 | 22,680 |
| V49 = T19+T10+T6+T4 | 5,040 | V84 = T21+T13+T8+T4 | 10,080 |
| V50 = T19+T10+T7+T4 | 5,040 | V85 = T21+T13+T8+T5 | 10,080 |
| V51 = T19+T10+T7+T5 | 7,560 | V86 = T22+T9+T6+T4 | 2,520 |
| V52 = T19+T12+T6+T4 | 2,520 | V87 = T22+T10+T6+T4 | 10,080 |
| V53 = T19+T12+T7+T4 | 5,040 | V88 = T22+T10+T7+T4 | 10,080 |
| V54 = T19+T12+T7+T5 | 7,560 | V89 = T22+T10+T7+T5 | 15,120 |
| V55 = T19+T12+T8+T4 | 7,560 | V90 = T22+T11+T6+T4 | 7,560 |
| V56 = T19+T12+T8+T5 | 7,560 | V91 = T22+T11+T7+T4 | 30,240 |
| V57 = T20+T9+T6+T4 | 5,040 | V92 = T22+T11+T7+T5 | 45,360 |
| V58 = T20+T10+T6+T4 | 15,120 | V93 = T22+T12+T6+T4 | 10,080 |
| V59 = T20+T10+T7+T4 | 15,120 | V94 = T22+T12+T7+T4 | 20,160 |
| V60 = T20+T10+T7+T5 | 22,680 | V95 = T22+T12+T7+T5 | 30,240 |
| V61 = T20+T11+T6+T4 | 5,040 | V96 = T22+T12+T8+T4 | 30,240 |
| V62 = T20+T11+T7+T4 | 20,160 | V97 = T22+T12+T8+T5 | 30,240 |
| V63 = T20+T11+T7+T5 | 30,240 | V98 = T22+T13+T6+T4 | 15,120 |
| V64 = T20+T12+T6+T4 | 10,080 | V99 = T22+T13+T7+T4 | 45,360 |
| V65 = T20+T12+T7+T4 | 20,160 | V100 = T22+T13+T7+T5 | 68,040 |
| V66 = T20+T12+T7+T5 | 30,240 | V101 = T22+T13+T8+T4 | 30,240 |
| V67 = T20+T12+T8+T4 | 30,240 | V102 = T22+T13+T8+T5 | 30,240 |

Table A.11. Tree-equivalent regular vines with 7 nodes (Continuation).

| Tree sequence & # Tree-equivalent Labeled Regular Vines | | Tree sequence & # Tree-equivalent Labeled Regular Vines | |
|--|--------|--|--------|
| V103 = T23+T9+T6+T4 | 5,040 | V136 = T24+T12+T8+T5 | 30,240 |
| V104 = T23+T10+T6+T4 | 10,080 | V137 = T24+T13+T6+T4 | 20,160 |
| V105 = T23+T10+T7+T4 | 10,080 | V138 = T24+T13+T7+T4 | 60,480 |
| V106 = T23+T10+T7+T5 | 15,120 | V139 = T24+T13+T7+T5 | 90,720 |
| V107 = T23+T11+T6+T4 | 5,040 | V140 = T24+T13+T8+T4 | 40,320 |
| V108 = T23+T11+T7+T4 | 20,160 | V141 = T24+T13+T8+T5 | 40,320 |
| V109 = T23+T11+T7+T5 | 30,240 | V142 = T24+T14+T6+T4 | 12,600 |
| V110 = T23+T12+T6+T4 | 5,040 | V143 = T24+T14+T7+T4 | 25,200 |
| V111 = T23+T12+T7+T4 | 10,080 | V144 = T24+T14+T7+T5 | 37,800 |
| V112 = T23+T12+T7+T5 | 15,120 | V145 = T24+T14+T8+T4 | 12,600 |
| V113 = T23+T12+T8+T4 | 15,120 | V146 = T24+T14+T8+T5 | 12,600 |
| V114 = T23+T12+T8+T5 | 15,120 | V147 = T25+T9+T6+T4 | 2,520 |
| V115 = T23+T13+T6+T4 | 20,160 | V148 = T25+T10+T6+T4 | 5,040 |
| V116 = T23+T13+T7+T4 | 60,480 | V149 = T25+T10+T7+T4 | 5,040 |
| V117 = T23+T13+T7+T5 | 90,720 | V150 = T25+T10+T7+T5 | 7,560 |
| V118 = T23+T13+T8+T4 | 40,320 | V151 = T25+T11+T6+T4 | 2,520 |
| V119 = T23+T13+T8+T5 | 40,320 | V152 = T25+T11+T7+T4 | 10,080 |
| V120 = T23+T14+T6+T4 | 25,200 | V153 = T25+T11+T7+T5 | 15,120 |
| V121 = T23+T14+T7+T4 | 50,400 | V154 = T25+T12+T6+T4 | 2,520 |
| V122 = T23+T14+T7+T5 | 75,600 | V155 = T25+T12+T7+T4 | 5,040 |
| V123 = T23+T14+T8+T4 | 25,200 | V156 = T25+T12+T7+T5 | 7,560 |
| V124 = T23+T14+T8+T5 | 25,200 | V157 = T25+T12+T8+T4 | 7,560 |
| V125 = T24+T9+T6+T4 | 5,040 | V158 = T25+T12+T8+T5 | 7,560 |
| V126 = T24+T10+T6+T4 | 15,120 | V159 = T25+T13+T6+T4 | 5,040 |
| V127 = T24+T10+T7+T4 | 15,120 | V160 = T25+T13+T7+T4 | 15,120 |
| V128 = T24+T10+T7+T5 | 22,680 | V161 = T25+T13+T7+T5 | 22,680 |
| V129 = T24+T11+T6+T4 | 7,560 | V162 = T25+T13+T8+T4 | 10,080 |
| V130 = T24+T11+T7+T4 | 30,240 | V163 = T25+T13+T8+T5 | 10,080 |
| V131 = T24+T11+T7+T5 | 45,360 | V164 = T25+T14+T6+T4 | 2,520 |
| V132 = T24+T12+T6+T4 | 10,080 | V165 = T25+T14+T7+T4 | 5,040 |
| V133 = T24+T12+T7+T4 | 20,160 | V166 = T25+T14+T7+T5 | 7,560 |
| V134 = T24+T12+T7+T5 | 30,240 | V167 = T25+T14+T8+T4 | 2,520 |
| V135 = T24+T12+T8+T4 | 30,240 | V168 = T25+T14+T8+T5 | 2,520 |