About The Number of Vines and Regular Vines on n Nodes.

Morales-Nápoles O.[*] Cooke R.M. & Kurowicka D.
Delft Institute of Applied Mathematics
Delft University of Technology
The Netherlands [†]

Abstract

The theory of graphs is important not only for combinatorial problems but in physics, electrical engineering, chemistry, social psychology, and research of operations. Labeled trees find application in probability theory. These objects were first successfully counted by Cayley in 1889. Vines generalize trees. They were introduced by Cooke in 1997 and they have been applied in uncertainty analysis. More recently applications in statistics have been developed in which distinguishing vines according to their graphical structure is of importance [1], [2], [3], [4], [5]. In this paper, previous results about the number of labeled trees on n nodes will be discussed. Some of the ideas previously used to characterize trees will be extended to characterize vines on n nodes. Algorithms to build vines, together with a result about the number of vines on n nodes will be presented.

1 Introduction

Men has always been fascinated by counting all sorts of different objects[1]. The problem of counting graphs has been undertaken in the past[8]. Trees find many applications in probability theory, decision analysis and optimization. These have also been the object of study for combinatorial problems and results have been presented regarding the number of labeled trees on n nodes.

Vines are graphical models that extend the idea of a tree [9]. These objects have found application in probability theory and uncertainty analysis (see for example [10]). More recently they are becoming popular in statistical analysis of data [1], [2], [3], [4], [5].

In this paper previous results concerning the number of trees on n nodes are briefly discussed in section 2. Section 3 presents two ways to characterize vines on n nodes. The first method

[*]Faculty of Electrical Engineering, Mathematics and Computer Science. Mekelweg 4 (HB06.160) 2628 CD Delft, the Netherlands. Tel: +31 1527 84563. Email: *o.moralesnapoles@ewi.tudelft.nl*

[†]The authors would like to thank Sandra Gáytan-Aguilar and Etienne de Klerk for their useful ideas for the completion of this paper.

[1]Calculating prodigies have counted many things along history, Jedediah Buxton (1702) an illiterate man from Elmton, England kept a mental record of all the free beer and ale he was given since the age of 12 and that averaged out to 5 or 6 ounces a day. When taken to see *Richard III* at the Drury Lane Playhouse in London "he declared after a fine piece of music, that the innumerable sounds produced by the instruments had perplexed him beyond measure, and he attended even to Mr. *Garrick* only to count the words that he uttered, in which, he says, he perfectly succeeded"[6]. Thomas Fuller, an African man shipped to America as a slave in 1724 "began his application to figures by counting to ten, and then when he was able to count a hundred, he thought himself (to use his own words) "a very clever fellow". His first attempt after this was to count the number of hairs in a cow's tail, which he found to be 2872"[7]

counts the total number of vines on n nodes and extracts the regular vines by discarding those vines which are non-regular. The second method constructs all possible regular vines on n nodes using line graphs at each level in the vine. Neither method yields the number of regular vines on n nodes as a function of n.

Section 4 characterizes regular vines as triangular arrays, and finds the number of regular vines on n nodes by extending a regular vine on $n-1$ nodes. This enables us to express the number of regular vines on n nodes as $\binom{n}{2} \times (n-2)! \times 2^{\binom{n-2}{2}}$. The results from section 3 may be contrasted with the result from section 4. For example, there are 11 unlabeled trees on 7 nodes each of which admits a number of regular vines. From these 11 trees, the one where every node has degree at most equal to 2 admits only one regular vine and can be labeled in 2,520 different ways. Other trees may be analyzed similarly to enumerate regular vines. In general for trees on seven nodes there are $1 \times 2,520 + 9 \times 2,520 + 19 \times 5,040 + 840 \times 33 + 630 \times 80 + 2,520 \times 168 + 840 \times 168 + 1,260 \times 342 + 420 \times 1,452 + 210 \times 29,28 + 7 \times 23,040 = 2,580,480 = \binom{7}{2} \times 5! \times 2^{\binom{7}{2}}$. Interestingly, the number of extensions of a regular vine on $n-1$ nodes to a regular vine on n nodes does not depend on the particular regular vine on $n-1$ nodes being extended. The final section gathers conclusions.

A graph will be denoted by $G = (E, N)$ where N is the node set and E the edge set of the graph which is a subset of pairs of N. Graphs in which individual nodes have no distinct identifications except through their interconnectedness are called unlabeled graphs. Without loss of generality in this presentation when $N = \{1, 2, ..., n\}$ we speak of **labeled graphs**.

2 Trees

A **tree** is an undirected acyclic graph. The graph isomorphism problem consist 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.

Definition 2.1. *Two* **labeled graphs** *$G_i = (E_i, N_i)$ and $G_j = (E_j, N_j)$ are* **isomorphic** *if there is a bijection $\varphi : N_i \to N_j$ 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.*

A connected graph $T = (N, E)$ is called a **labeled tree** *with nodes $N = \{1, 2, ..., n\}$ and edges E, where E is a subset of pairs of N with no cycle.*

In this section labeled trees will be briefly discussed. These structures have been used to represent high dimensional probability distributions [9] and they are often called dependence trees. This section however will be concerned with the properties of trees only as graphs. For an account of dependence trees see [11]. We begin our presentation with a well known result about trees.

2.1 The Number of Labeled Trees on n Nodes and the Prüfer Code

Two different labeled trees on 5 nodes are presented in figures 2.1 and 2.2. The reader may observe that permuting nodes 1 and 5 in T_1 transforms it into T_2 and hence they would be considered the same unlabeled tree. In this section the interest will be mainly in labeled trees.

The first proof about the number of labeled trees on n nodes is due to Cayley in 1889 [12]. Since then several proofs have been presented [13].

Theorem 2.1. *The number of labeled trees on n nodes is n^{n-2}.*

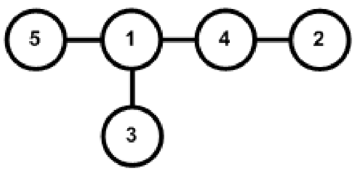


Figure 2.1: T_1 a tree on 5 nodes.

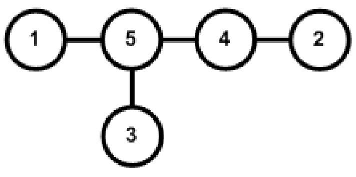


Figure 2.2: T_2 a tree on 5 nodes.

One of various proofs due to Prüfer in 1918 [14] 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 ordered $(n-2)$-tuples $(A_1, A_2, ..., A_{n-2})$ where each A_i is an integer not greater than n.

Definition 2.2. *Every sequence of numbers $R(T_k) = (A_1, A_2, ..., A_{n-2})$ where each A_i is an integer not greater than n is a* **Prüfer Code** *for some tree T_k on n nodes.*

In his paper Prüfer obtains the correspondence by the following procedure: For a given tree, remove the endpoint[2] with the smallest label (other than the root[3]) 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 2.1 and 2.2 have Prüfer codes $R(T_1) = (4, 1, 1)$ and $R(T_2) = (5, 4, 5)$ respectively. The procedure described above may be easily reversed, that is, suppose you start with a sequence of $(n-2)$-tuples $R(T_k) = (A_1, A_2, ..., A_{n-2})$ then to obtain the only tree corresponding to the sequence one applies algorithm 2.1:

Algorithm 2.1. *Decoding a Prüfer code.*

1. Take a sequence $R(T_k) = (A_1, A_2, ..., A_{n-2})$ for $k = 1, 2, .., n^{n-2}$ where each A_i, $i = 1, 2, ..., n-2$ 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.
$S(T_k) = \begin{pmatrix} A_1 & A_2 & A_3 & ... & n \\ B_1 & B_2 & B_3 & ... & B_{n-1} \end{pmatrix}$

Take the two Prüfer codes $R(T_1) = (4, 1, 1)$ and $R(T_2) = (5, 4, 5)$. Apply algorithm 2.1 to decode each sequence into the extended Prüfer code. The reader may check in equation 2.1 that $S(T_1)$ corresponds to figure 2.1 and $S(T_2)$ to figure 2.2.

[2]The endpoints are nodes with degree one in the tree, they are sometimes referred to as *leafs*.

[3]Without 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.

$$S(T_1) = \begin{pmatrix} 4 & 1 & 1 & 5 \\ 2 & 3 & 4 & 1 \end{pmatrix}, S(T_2) = \begin{pmatrix} 5 & 4 & 5 & 5 \\ 1 & 2 & 3 & 4 \end{pmatrix} \tag{2.1}$$

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. Labeled trees are interesting not only as objects that can be counted and subject of combinatorial problems. They find application in optimization, probability theory and uncertainty analysis ([9], [11]). In next section vines will be discussed and the ideas presented in this section will be extended to deal with these graphical objects.

3 Vines

A vine [9] is a set of nested trees. Just as labeled trees, vines have been used to represent high dimensional probability distributions [15] and [11] with applications in uncertainty analysis. More recently they are being applied in statistical analysis of multivariate data sets [1], [3], [2] and [5]. These last references are concerned with choosing an optimal vine to represent multivariate data sets. Algorithms for enumerating all possible regular vines on n nodes will be needed for this purpose. All trees in a vine may be thought of as labeled trees. In this section some results about the number of vines on n nodes will be presented.

3.1 The Number of Vines on n Nodes and the Prüfer Code

The ideas presented in section 2.1 can be extended to count the number of vines (and regular vines) that are possible on n nodes. This will be shown in the present subsection. The presentation begins with the definitions of vine and regular vine.

Definition 3.1. *$V(n)$ is a* **labeled vine** *on n elements if:*

1. *$V(n) = (T_1, T_2, T_3, T_4, ..., T_n)$.*
2. *T_1 is a labeled tree with nodes $N_1 = 1, 2, ..., n$ and edges E_1. For $i = 2, ..., n$, T_i is a labeled tree with nodes $N_i = E_{i-1}$. E_{i-1} has been given a unique labeling.*

If in addition for $i = 2, ..., n-1$, if $(a, b) \in E_i$, then $|a \triangle b| = 2$, where $\triangle$ denotes the symmetric difference, then $V(n)$ is a **labeled regular vine**. *In other words, if a and b are nodes of T_i connected by an edge in T_i, where $a = \{a_1, a_2\}$ and $b = \{b_1, b_2\}$, then exactly one of the a_i equals one of the b_i. This condition is called the* **proximity** *condition.*

The 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**. Formal definitions may be found in [11]. Vines (and regular vines) may be classified according to the unlabeled tree used at each level in the vine. For this reason the following definition is introduced.

Definition 3.2. *If a bijection as in definition 2.1 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.*

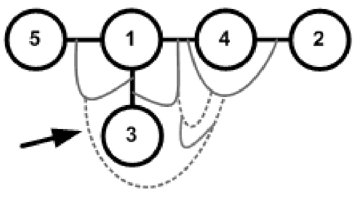


Figure 3.1: Non-regular vine on 5 nodes.

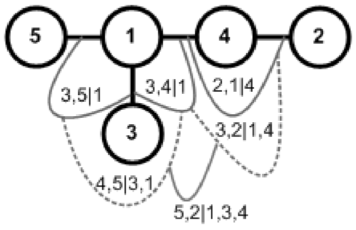


Figure 3.2: Regular vine on 5 nodes.

In figures 3.1 and 3.2 respectively a non-regular and a regular vine on five nodes are generated. The edge that makes figure 3.1 a non-regular vine is indicated by an arrow. The conditioned set is separated from the conditioning set by a vertical line "|" in figure 3.2. Obviously these two vines are different labeled vines. However, according to definition 3.2 they are the same tree-equivalent vine. Observe that by permuting the numbers in T_1 in figure 3.2 we would generate different labeled regular vines but according to definition 3.2 the same tree-equivalent vine.

Since every labeled tree can be represented by a Prüfer code, then every sub-tree in the 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 3.1.

Algorithm 3.1. *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):
   (i) 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 [4].
   (ii) 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 $n - 1^{th}$ tree. At this point there is only one way to connect them and no Prüfer code is required.

From algorithm 3.1 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 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 3.1 and $V_2(5)$ the one in 3.2 :

$$V_1(5) = \begin{pmatrix} 4 & 1 & 1 \\ & 3 & 2 \\ & & 1 \end{pmatrix}, V_2(5) = \begin{pmatrix} 4 & 1 & 1 \\ & 3 & 2 \\ & & 2 \end{pmatrix} \tag{3.1}$$

[4]This 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.

Corollary 3.1. *The number of vines on n nodes is* $\prod_{i=1}^{n} i^{i-2}$.

Proof. The proof is in fact algorithm 3.1. This is a consequence of theorem 2.1 and definition 3.1.□

Regular vines are most interesting in uncertainty analysis. Implementing Algorithm 3.1 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 (see table 1). According to corollary 3.1 the number of vines grows extremely fast with n and it could be very restrictive in time to find all regular vines even for a modest number of nodes (8 or 9). Another possibility to construct only regular vines will be discussed in the next subsection.

3.2 Regular vines and the line graph

As stated at the end of previous section, another possibility is available to produce only regular vines as opposed to producing all possible vines and discarding those that are not regular as in algorithm 3.1. The idea is to use the line graph[5] of each tree in the vine. Harary notes in [17] that the concept of the line graph of a given graph is so natural that is has been rediscovered independently by many authors.

Definition 3.3. *[16] 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.*

If the edges of the first tree of figure 3.2 are labeled according to the second step in algorithm 3.1 then the line graph of this tree can be found according to definition 3.3. This line graph corresponds to figure 3.3. Nodes 1, 2, 3 and 4 in figure 3.3 corresponds to edges (4,1), (1,3), (1,4) and (5,1) respectively in figure 3.2.

If in the same way we label the nodes of the second tree in the vine in figure 3.2 accordingly, then the line graph in figure 3.4 may be obtained. In this new line graph, nodes 1, 2, 3 correspond respectively to nodes $(2, 1|4)$, $(3, 4|2)$ and $(3, 5|1)$ in figure 3.2.

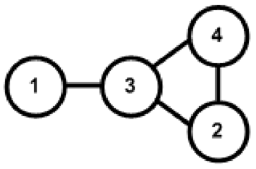


Figure 3.3: Line Graph of the first tree in figure 3.2

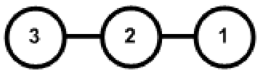


Figure 3.4: Line Graph of the second tree of the vine from figure 3.2.

Definition 3.4. *[18] A* **spanning subgraph** *T of a graph G is a subgraph with the same set of nodes as G. If T is a tree, it is called a* **spanning tree** *of G.*

It is clear from definitions 3.3 and 3.4 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 most be found. This result is summarized in algorithm 3.2.

Algorithm 3.2. *Constructing all possible regular vines on n nodes.*

[5]Line graphs are also known as derived graphs, interchange graphs, adjoint and edge to vertex dual[16].

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):
   (i) 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 [6].
4. Construct the line graph of each one of the trees from step 2.
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 $n - 1^{th}$ 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 3.1 once a way of labeling the edges from each tree in the vine is specified. Algorithm 3.2 does not produce any irregular vine as opposed to algorithm 3.1. 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 [7] is demanding in terms of time and space [22].

Table 1 presents a summary with the number of labeled trees, vines and regular vines on 3, 4, 5, 6, 7 and 8 nodes[8]. The second column presents the number of unlabeled trees on n nodes. The third column corresponds to the values obtained by applying the formula in theorem 2.1 and the fourth to values obtained by applying the formula in corollary 3.1. Algorithms 3.1 and 3.2 allow to count the number of regular vines on n nodes. The number of regular vines on up to 7 nodes was found using algorithm 2.1 and the values for 8 nodes using algorithm 3.2[9]. The results of counting regular vines with algorithms 2.1 and 3.2 are presented in column 5. To implement algorithm 3.2, MATGRAPH [24] was used to find line graphs for each of the 23 unlabeled trees on 8 nodes. A version of the Mayeda-Seshu algorithm was used [22] to find all spanning trees of each of the 23 line graphs.

Column six in table 1 presents the number of tree-equivalent regular vines on n nodes. Algorithm 3.1 may be used to list the number of tree-equivalent vines (or tree-equivalent regular vines) on n nodes by checking for isomorphism at each level in the vine. Also, algorithm 3.2 can be used to count the number of tree-equivalent regular vines on n nodes by checking tree isomorphism at each level of the vine[10]. All possible tree-equivalent vines with at most 5 nodes in T_1 are presented in figure A.1 in appendix 1.

[6]As 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.

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

[8]For 1 and 2 nodes there is exactly one of each object.

[9]Actually algorithm 3.2 does not need to be implemented completely to count the number of regular vines on 8 nodes. Observe that it is sufficient to know how many spanning tress of each unlabeled class in $n-1$ nodes does a line graph of a tree in n nodes contain.

[10]As for counting regular vines algorithms 3.1 and 3.2 do not need to be implemented completely to count the number of tree-equivalent regular vines on 8 and 9 nodes. Observe that it is sufficient to know how many spanning tress of each unlabeled class in $n-1$ nodes does a line graph of a tree in n nodes contain.

Nodes	Trees		Vines		
	A[a]	B[b]	C[c]	D[d]	E[e]
3	1	3	3	3	1
4	2	16	48	24	2
5	3	125	6,000	480	5
6	6	1,249	7,776,000	23,040	22
7	11	16,807	130,691,232,000	2,580,480	136
8	23	262,144	34,259,922,321,408,000	660,602,880	1,464

Table 1: Number of unlabeled and labeled trees, vines, regular vines and tree-equivalent vines in 3, 4, 5, 6, 7 and 8 nodes.

[a]Number of unlabeled trees
[b]Number of labeled trees
[c]Number of labeled vines
[d]Number of labeled regular vines
[e]Number of tree-equivalent regular vines

Table 2 presents a catalogue with non-isomorphic trees on 1, 2, 3, 4, 5, 6 and 7 nodes and some relevant characteristics of each one. A similar catalogue was presented in [13] for trees with at most five nodes. In [25] a catalogue of non-isomorphic tress with at most 8 nodes may be found[11]. None of the above catalogues presents results for vines. An extended catalogue for non-isomorphic trees with up to 9 nodes similar to the one in table 2 is available from the authors on request.

The concept of the line graph also allows to obtain bounds for the number of regular vines admissible by unlabeled trees on n nodes. These results are presented next as lemmas. Lemma 3.2 that is rather evident has been stated in [9] without a proof.

Lemma 3.1. *If the first tree of a vine on n nodes has one node with maximal degree, then the number of labeled regular vines possible with this tree equals the number of labeled regular vines on $(n-1)$ nodes.*

Proof. Since every edge in T_1 is adjacent to each other then the line graph of this tree is a complete graph on $(n-1)$ nodes that has $(n-1)^{n-3}$ possible spanning trees. These are all possible labeled trees on $n-1$ nodes each of which admits a fixed number of labeled regular vines.□

Lemma 3.2. *If the first tree of a vine on n nodes has $(n-2)$ nodes with degree 2, then the number of regular vines possible with this tree equals 1.*

Proof. Observe that the line graph of T_1 will be also a tree on $n-1$ nodes with $(n-3)$ nodes with degree 2. Hence its only possible spanning tree will be itself and to preserve regularity this tree should be used in T_2. The same argument holds for all $j \geq 2$ and hence only one regular vine is possible.□

Lemma 3.2 provides a lower bound for the number of regular vines possible for a given unlabeled tree T_1. In the same way lemma 3.1 provides an upper bound. This result may be observed in table 2. A more general result for counting labeled regular vines is dealt with in next section.

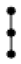
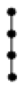
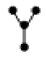
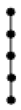
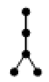
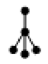
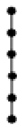
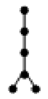
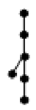
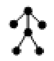
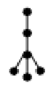
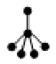
[11]This catalogue repeats a tree in eight nodes neglecting another one. In the same reference tables counting the number of rooted trees on up to 26 nodes and the number of non-isomorphic trees on less than 26 nodes may be found.

In applications two kind of regular vines have been most widely used. ***C-Vines*** are regular vines for which each tree in the vine has one node with maximal degree. ***D-Vines*** are regular vines for which the first tree of the vine has $(n-2)$ nodes with degree 2. Next results about the number of D-vines and C-vines on n nodes are presented. Both results where presented in [1] with proofs that are slightly different to the ones presented here.

Lemma 3.3. *The number of* C-vines *on n nodes equals the number of* D-vines *on n nodes and is* $\frac{n!}{2}$

Proof. For C-vines observe that there are n possible labeled trees on n nodes for which a single node has maximal degree. Once the first tree has been fixed any of the $(n-1)$ edges may be chosen so as to construct any of the $(n-1)$ possible labeled trees on $(n-1)$ nodes for which a single node has maximal degree. Any of these would preserve regularity. The same argument holds for all other trees on the vine until two edges need to be connected as nodes in T_{n-1}. Hence there are $n \cdot (n-1) \cdot (n-2) \cdot ... \cdot (3) = \frac{n!}{2}$ C-vines on n nodes.

For D-vines observe that from lemma 3.2, $T_1 \in V$ completely determines the vine. And since there are $\frac{n!}{2}$ ways of choosing it the result follows. □

Prüfer code example			1 	12 	11 	123 	112 	111 	1234 	1123 	1213 	2244 	1112 	1111 
	Non-isomorphic trees													
# Labeled Trees	1	1	3	12	4	60	60	5	360	360	360	90	120	6
# Regular Vines on each tree	1	1	1	1	3	1	5	24	1	7	11	48	75	480

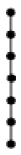
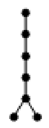
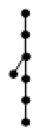
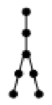
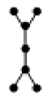
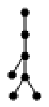
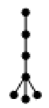
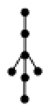
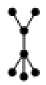
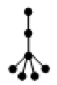
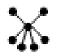
Prüfer code example	12345 	12344 	12234 	12324 	11233 	11223 	11123 	12223 	11122 	11112 	11111 
	Non-isomorphic trees										
# Labeled Trees	2,520	2,520	5,040	840	630	2,520	840	1,260	420	210	7
# Regular Vines on each tree	1	9	19	33	80	168	168	342	1,452	2,928	23,040

Table 2: Catalogue of non-isomorphic trees with up to 7 nodes.

4 The Number of Regular Vines on n Nodes.

So far the number of vines has been obtained from Cayley's theorem in corollary 3.1. Results concerning the number of tree-equivalent and labeled regular vines on at most 8 nodes have been presented by using Prüfer codes and line graphs. This section derives a formula for the number of regular vines on n nodes.

Definition 4.1. *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: $e \in e_1 \in ... \in f$, we say that e is an* **m-descendent** *of f.*

Lemma 4.1. *[11] For any node M of order $k > 0$ in a regular vine, if node i is a member of the conditioned set of M, then i is a member of the conditioned set of exactly one of the m-children of M, and the conditioning set of an m-child of M is a subset of the conditioning set of M.*

Definition 4.2. *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.*

Regularity[12] means that every node in T_i, $i \geq n-1$ must have a sibling and a common child with its sibling. In this section, another triangular array representing a regular vine will be introduced. One disadvantage of using a triangular array such as the one used in section 3.1 is that the information regarding the label of nodes 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. The idea of the construction presented here is to preserve the information concerning the labels of the first tree as lower trees in the vine are added. In analogy to a Prüfer code a sequence of n-tuples $(A_n, A_{n-1}, ..., A_1)$ where each A_i is an integer not greater than n will be called a **natural order**. This is defined next.

Definition 4.3. *A* **natural order** *of the elements of a regular vine $V(n)$ on n elements is a sequence of numbers $NO(V(n)) = (A_n, A_{n-1}, ..., A_1)$ where each A_i is an integer not greater than n obtained as follows: Take one conditioned element of the last tree of a regular vine (a tree with a single node and no edges) and assign it position n; assign the other conditioned element of the top node position $(n-1)$. Element A_{n-1} occurs in one m-child of the top node with an $(n-1)$-partner in the conditioned set. Give this $(n-1)$-partner position $(n-2)$. The $(n-2)$ partner of element A_{n-2} is assigned position $(n-3)$. Iterate this process until all elements have been assigned a position.*

Observe that there are two natural orders for every regular vine. A representation of the regular vine in figure 3.2 using a directed graph is presented in figure 4.1. This representation will be useful in the rest of the chapter for explaining some of the concepts introduced. The nodes of each tree in the regular vine are nodes in the directed graph. Observe that every parent node has exactly two children. The conditioned set is presented to the left of a vertical line (| sign) and the conditioning set to its right.

The element in position n occurs as conditioned variable in tree T_n (this tree has one node and no edges). The element in position $(n-j)$ occurs in the unique node of tree T_{n-j} with conditioned set $\{A_{n-(j+1)}, A_{n-j}\}$. If 5 is chosen as A_n, then by definition 4.3 the natural order of the regular vine would be $NO_1(V_2(5)) = (5, 2, 3, 4, 1)$. In the same way if node 2 was chosen as element A_n then the natural order would be $NO_2(V_2(5)) = (2, 5, 4, 3, 1)$. A regular vine may be coded as a lower triangular array with the natural ordering on the diagonal. The natural

[12]Or proximity in the language of section 3

order will be used in a triangular array similar to the one introduced in section 3.1 but that preserves all the information needed regarding conditioned and conditioning sets in the regular vine. In particular, the regular vine array defined below encodes all the information in figure 4.1 using single digits in each cell of the array.

$$\begin{tikzcd}
5,2|1,3,4 \arrow[d] \arrow[dr] & & & & \\
4,5|3,1 \arrow[d] \arrow[drr] & 3,2|1,4 \arrow[d] \arrow[dr] & & & \\
3,5|1 \arrow[d] \arrow[drr] & 1,2|4 \arrow[d] \arrow[drr] & 3,4|1 \arrow[d] \arrow[dr] & & \\
1,5 \arrow[d] \arrow[drrrr] & 4,2 \arrow[d] \arrow[drr] & 1,3 \arrow[d] \arrow[drr] & 1,4 \arrow[d] \arrow[dr] & \\
5 & 2 & 3 & 4 & 1
\end{tikzcd}$$

Figure 4.1: Representation of the regular vine in figure 3.2

Definition 4.4. *A* ***regular vine array TA(V(n))*** $= \{A_{i,j}\}$ *for* $i, j = 1, ..., n$ *and* $j \geq i$ *is a lower triangular matrix with elements in* $\{1, ..., n\}$ *indexed in 'reverse order' (see equation 4.1), where* $A_{j,j}$ *equals the element in position* j *in* $NO(V(n))$ *and* $A_{j-1,j}$ *equals the element in position* $j-1$ *in the same natural order. The* **echelon** *of element* $A_{i,j}$ *is* i *and element* $A_{i,j}$ *codes the node* $(A_{j,j}, A_{i,j}|A_{i-1,j}, ..., A_{1,j})$

The regular vine array $TA(V_2(5))$ corresponding to figure 4.1 using $NO_1(V_2(5))$ is presented in equation 4.1. Observe that the row and column indices are in their usual position but their sense is reversed (with respect to traditional matrix indexing) in order to facilitate adding new variables to the left. From definition 4.4, we may speak unambiguously of "node", "element" or "variable" $A_{i,j}$. Thus the "node $A_{i,j}$" is the set of elements "$(A_{i,j}, A_{j,j}|A_{i-1,j}, \ldots A_{1,j})$", arranged to separate the conditioned elements from the conditioning elements by "|".

$$TA(V_2(5)) = \begin{pmatrix} A_{5,5} & & & & \\ A_{4,5} & A_{4,4} & & & \\ A_{3,5} & A_{3,4} & A_{3,3} & & \\ A_{2,5} & A_{2,4} & A_{2,3} & A_{2,2} & \\ A_{1,5} & A_{1,4} & A_{1,3} & A_{1,2} & A_{1,1} \end{pmatrix} = \begin{pmatrix} 5 & & & & \\ 2 & 2 & & & \\ 4 & 3 & 3 & & \\ 3 & 1 & 4 & 4 & \\ 1 & 4 & 1 & 1 & 1 \end{pmatrix} \tag{4.1}$$

From figure 4.1 and equation 4.1 it may be observed that a regular vine may be represented by a regular vine array as described in definition 4.4, in which the nodes of each tree in a regular vine have children in the immediate lower order tree. Conditions for child nodes in the regular vine array are given next.

Definition 4.5. *Node* $A_{i-1,h}$ *is a* **child** *of node* $A_{i,j}$ *if:*

(i) $\{A_{h,h}, A_{i-1,h}, A_{i-2,h}, ..., A_{1,h}\} \subset \{A_{j,j}, A_{i,j}, A_{i-1,j}, A_{i-2,j}, ..., A_{1,j}\}$

(ii) $|\{A_{h,h}, A_{i-1,h}, A_{i-2,h}, ..., A_{1,h}\}| = |\{A_{j,j}, A_{i,j}, A_{i-1,j}, A_{i-2,j}, ..., A_{1,j}\}| - 1$

(iii) $|\{A_{h,h}, A_{i-2,h}\} \cap \{A_{j,j}, A_{i-1,j}\}| = 1$

The reader may check for example that according to definition 4.5 $A_{2,4} = (2,1|4)$ and $A_{2,3} = (3,4|1)$ in 4.1 are children of $A_{3,4} = (2,3|1,4)$. According to definition 4.2, $A_{3,4} = (2,3|1,4)$ and $A_{3,5} = (5,4|3,1)$ are siblings because they are children of the common parent $A_{4,5} = (5,2|4,3,1)$. Similarly $A_{2,3} = (3,4|1)$ and $A_{2,5} = (5,3|1)$ are children of $A_{3,5} = (5,4|3,1)$ and hence siblings. Other elements may be also checked by the reader. Next it will be shown that a matrix such as the one in definition 4.4 represents a regular vine.

We characterize first those regular vine arrays which represent regular vines.

Theorem 4.1. *$TA(V(n))$ represents a regular vine $\Longleftrightarrow$ $TA(V(n))$ satisfies **condition R**. That is, for all $i \geq 2$, element $A_{i,j} = A_{h,h}$ or $A_{i,j} = A_{i-1,h}$ for some h such that $i \leq h < j$ and $\{A_{j,j}, ..., A_{i+1,j}\} \cap \{A_{i-1,h}, ..., A_{1,h}\} = \varnothing$*

Proof. $\Rightarrow$ If $V(n)$ is a regular vine then every node $A_{i,j}$ in $TA(n)$ has two children in echelon $i-1$ one of which is $A_{i-1,j}$. Suppose the other child is in column h, then condition R follows from $(i), (ii), (iii)$ in definition 4.5.

$\Leftarrow$ Let $TA(k)$ be a regular vine array satisfying condition R. If $k = 3$, the nodes of $TA(k)$ clearly satisfy regularity. Suppose the theorem holds for $k = n-1$. Node $A_{n-1,n}$ satisfies regularity by definition 4.4. An induction will show that nodes $A_{n-1,n}, ..., A_{1,n}$ satisfy regularity. We show first that node $A_{n-2,n}$ has a sibling and has a common child with this sibling. By condition R element $A_{n-2,n}$ is equal to element $A_{n-2,n-2}$ or $A_{n-3,n-2}$. In either case, node $A_{n-3,n-2}$ is a child of node $A_{n-2,n}$ and hence node $A_{n-2,n}$ satisfies regularity.

Suppose that for every $j = n-2, ..., k+1$, every node $A_{j,n}$, satisfies regularity. We claim that $A_{k,n}$ must also satisfy regularity.

Node $A_{k,n}$ is a child of node $A_{k+1,n}$ and by the induction hypothesis, node $A_{k+1,n}$ satisfies regularity, therefore, it has a second child node $A_{k,h}$ and relation 4.2 must hold according to condition R.

$$\{A_{h,h}, A_{k,h}, A_{k-1,h}, ..., A_{1,h}\} = \{A_{k+1,n}, A_{k,n}, ..., A_{1,n}\} \tag{4.2}$$

Two situations are possible:

(i) $A_{k+1,n} = A_{h,h}$ or,

(ii) $A_{k+1,n} = A_{k,h}$

By induction node $A_{k,h}$ has two children, one of which is node $A_{k-1,h}$. It will be shown that one of these children must be a child of node $A_{k,n}$. In other words it will be shown that $A_{k,n}$ and $A_{k,h}$ are siblings and have a common child which is the condition for regularity.

In case (i) node $A_{k-1,h}$ cannot be a child of node $A_{k,n}$ since node $A_{k-1,h}$ contains element $A_{h,h} = A_{k+1,n}$ in its conditioned set, and element $A_{k+1,n}$ cannot belong to the constraint set of node $A_{k,n}$. The other child of node $A_{k,h}$ must be node $A_{k-1,m}$ for some $k \leq m < h$. This child cannot contain element $A_{h,h}$, and:

$$\{A_{m,m}, A_{k-1,m}, ..., A_{1,m}\} = \{A_{k,h}, A_{k-1,h}, ..., A_{1,h}\} \tag{4.3}$$

By induction, node $A_{k,h}$ satisfies regularity; therefore either element $A_{k,h}$ = element $A_{m,m}$ or element $A_{k,h}$ = element $A_{k-1,m}$, in either case by combining 4.2 and 4.3 we see that node $A_{k-1,m}$ is a child of node $A_{k,n}$.

In case (ii) element $A_{k+1,n} \neq$ element $A_{h,h}$, and equation 4.2 must still hold and by induction $A_{k,n} = A_{h,h}$ or $A_{k,n} = A_{k-1,h}$; in either case node $A_{k-1,h}$ will be a child of node $A_{k,n}$ and the latter will satisfy regularity.□

We now count the number of ways of extending an $n-1$ regular vine with a fixed natural ordering. This is equivalent to adding an additional column to the left of a regular vine in the regular vine array.

Evidently the top two elements of this new column are fixed, and the last element is fixed by the choices for the elements above it. If there are n elements in the new column, there are $n-3$ elements to be chosen. It will be seen that the number of extensions is in fact 2^{n-3} regardless of the regular vine being extended.

Theorem 4.2. *For any vine on $n-1$ elements, the number of regular vines on n nodes which extend this vine, preserving the natural ordering of the $n-1$ vine is 2^{n-3}.*

Proof. Let $V(n-1)$ be an arbitrary regular vine on $n-1$ elements with a natural order and $TA(V(n-1))$ its regular vine array. $TA(V(n-1))$ will be extended by adding a column of n elements to the left whose top two entries are $A_{n,n}$, $A_{n-1,n}$.The goal is to count the number of ways of adding a column to the left of $TA(V(n-1))$, so as to preserve regularity. Node $A_{k,n}$ satisfies regularity if it has a sibling which is a child of node $A_{k+1,n}$ and has a child which is also a child of its sibling. This latter child must be a node in $V(n-1)$. If each node $A_{k,n}$ for $k = 2, ..., n-2$, satisfies regularity, then $TA(V(n))$ (the extended regular vine array) codes a regular vine which extends the original regular vine $V(n-1)$.

$V(n-1)$ has trees $T_{n-1}, ..., T_1$ where T_{n-1} has one node and no edges, T_1 has $n-1$ nodes and $n-2$ edges; in general for $j = 1, ..., n-1$ tree T_{n-j} has j nodes and $j-1$ edges. After adding node $A_{n,n}$, T_1 will have n nodes and $n-1$ edges, T_2 will have $n-1$ nodes and $n-2$ edges and so on until tree n that will have a single node $A_{n-1,n} = (A_{n,n}, A_{n-1,n}|A_{n-2,n}, ..., A_{1,n})$ This node must have two children. One child must be, evidently, node $A_{n-2,n} = (A_{n,n}, A_{n-2,n}|A_{n-3,n}, ..., A_{1,n})$ and the other is the top node of $V(n-1)$ which is $A_{n-2,n-1} = (A_{n-1,n-1}, A_{n-1,n-2}|A_{n-3,n-1}, ..., A_{1,n-1})$. To satisfy regularity, nodes $A_{n-2,n}$ and $A_{n-2,n-1}$ must have a common child. This common child cannot contain element $A_{n-1,n} = A_{n-1,n-1}$ since it does not belong to node $A_{n-2,n}$ and hence the child must be of the form:

$$(A_{n-3,n-2}, A_{n-2,n-2}|A_{n-4,n-2}, ..., A_{1,n-2}).$$

The situation is pictured in figure 4.2.

$$\begin{tikzcd}
A_{n-1,n} \arrow[d] \arrow[dr] & & \\
A_{n-2,n} \arrow[d] \arrow[drr] & A_{n-2,n-1} \arrow[d] \arrow[dr] & \\
A_{n-3,n} & (A_{n-3,n-1}, A_{n-1,n-1}| A_{n-4,n-1}, ..., A_{1,n-1}) & (A_{n-3,n-2}, A_{n-2,n-2}| A_{n-4,n-2}, ..., A_{1,n-2})
\end{tikzcd}$$

Figure 4.2: Regular Vine Growing

Since element $A_{n-2,n}$ must be in exactly one of the children of the node $A_{n-2,n-1}$ it follows that element $A_{n-2,n}$ must be element $A_{n-2,n-2}$ or element $A_{n-3,n-2}$ either choice satisfying regularity.

Assume that elements $A_{n-1,n}, ..., A_{k+1,n}$ satisfying regularity have been found. We show that variable $A_{k,n}$ can be found such that node $A_{k,n}$ satisfies regularity, and that there are exactly two choices for this element. Node $A_{k+1,n}$ may be written $A_{k+1,n} = (A_{n,n}, a|b, c, d, ..., e)$ with children as in figure 4.3 below:

$$\begin{tikzcd}
A_{k+1,n} \arrow[d] \arrow[dr] & & \\
(A_{n,n}, X|\{b, c, d, ..., e\} \setminus X) & \begin{matrix}(a, f|\{b, c, d, ..., e\} \setminus f);\\ f \in \{b, c, d, ..., e\}\end{matrix} \arrow[d] \arrow[dr] & \\
& \begin{matrix}(f, g|\{\{b, c, d, ..., e\} \setminus f\} \setminus g);\\ g \in \{b, c, d, ..., e\} \setminus f\end{matrix} & \begin{matrix}(a, h|\{\{b, c, d, ..., e\} \setminus f\} \setminus h);\\ h \in \{b, c, d, ..., e\} \setminus f\end{matrix}
\end{tikzcd}$$

Figure 4.3: Regular Vine Growing

Node $(a, f|\{b, c, d, ..., e\} \setminus f)$ exists in the original vine $V(n-1)$ by assumption. Node $(A_{n,n}, X|\{b, c, d, ..., e\} \setminus X)$ satisfies regularity if $X = f$ or $X = g$, either choice being possible. No other choice is possible, as no other node can have constraint set $\{b, c, d, ..., e\}$. Note that if $k = 2$ then $\{\{b, c, d, ..., e\} \setminus f\} \setminus g) = \{\{b, c, d, ..., e\} \setminus f\} \setminus h) = \varnothing$.

It follows that for each node $A_{n-2,n}, ..., A_{2,n}$ there is a choice among 2 alternatives. Hence there are 2^{n-3} extensions of $V(n-1)$ to a regular vine on n elements.□

For the example from figure 4.1 the 2^{6-3} possible extensions of the regular vine array from example 4.1 are given by the tree in figure 4.4 bellow. Corollary 4.1 follows immediately from theorem 4.2.

$$\begin{tikzcd}
&&&&&&& 5 \arrow[dllll] \arrow[drrrr] &&&&&&& \\
&&& 3 \arrow[dll] \arrow[drr] &&&&&&&& 2 \arrow[dll] \arrow[drr] &&& \\
& 2 \arrow[dl] \arrow[dr] &&&& 1 \arrow[dl] \arrow[dr] &&&& 4 \arrow[dl] \arrow[dr] &&&& 3 \arrow[dl] \arrow[dr] & \\
1 && 4 && 4 && 2 && 1 && 3 && 1 && 4
\end{tikzcd}$$

Figure 4.4: 8 Possible Extensions of the $TA(V_2(5))$in equation 4.1 Representing the Vine in Figure 3.2

Corollary 4.1. *The number of regular vines possible with a fixed natural order $NO(n) = A_{n,n}, A_{n-1,n-1}, ..., A_{1,1}$ is:* $\prod_{j=1}^{n-3} 2^j = 2^{\binom{n-2}{2}}$

Proof. Start with a regular vine on three nodes with an arbitrary natural order and extend it to a regular vine on four nodes. Elements $A_{4,4}$, $A_{3,4}$ and $A_{1,4}$ are fixed by the natural order and hence only element $A_{2,4}$ may be chosen in 2 distinct ways. For each one of the 2 choices of $A_{2,4}$, from theorem 4.2 an extension to a regular vine on 5 nodes leaves two choices for each of the two elements $A_{3,5}$ and $A_{2,5}$. Continue this way until a regular vine on n nodes is formed and the result follows. □

Observe that corollary 4.1 implies that no regular vine would be counted twice once the natural order has been fixed. Obviously two regular vine arrays that are equal will represent the same vine. Once the number of regular vines that may be obtained with a given natural order is known, all that is left to know the number of regular vines on n nodes is how many natural orders are possible in order to produce all possible regular vines. Corollary 4.2 completes the problem of enumerating regular vines.

Corollary 4.2. *There are* $\binom{n}{2} \times (n-2)! \times 2^{\binom{n-2}{2}}$ *labeled regular vines in total.*

Proof. 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 $A_{n-2,n-2}, .., A_{1,1}$. By corollary 4.1 the proof is completed. □

The results of corollary 4.2 may be observed in tables 1 and 2 which were obtained by the methods explained in previous sections. For example, the number of regular vines on 7 nodes is $\binom{7}{2} \times 5! \times 2^{\binom{7}{2}} = 2520 + 9 \times 2520 + 19 \times 5040 + 840 \times 33 + 630 \times 80 + 2520 \times 168 + 840 \times 168 + 1260 \times 342 + 420 \times 1452 + 210 \times 2928 + 7 \times 23040 = 2,580,480$.

Remark 4.1. *By lemma 3.1 and corollary 4.2 it may be seen that a tree with a single node with maximum degree admits* $\binom{n-1}{2} \times (n-3)! \times 2^{\binom{n-3}{2}}$ *regular vines.*

Finally, the results of remark 4.1 may be also observed in table 2. For example there are 7 trees with maximal degree on 7 nodes each of which admits $\binom{6}{2} \times (4)! \times 2^{\binom{4}{2}} = 360 + 7 \times 360 + 11 \times 360 + 48 \times 90 + 75 \times 120 + 6 \times 480 = 23,040$ regular vines which is exactly the total number of regular vines on 6 nodes. To finalize some conclusions are presented next.

5 Conclusion

This paper investigates counting problems related to vines. Corollary 3.1 has been obtained from Cayley's theorem 2.1 to count the number of vines on n nodes. 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 2.1 is easy to implement and efficient if regular vines on less than 6 *nodes* are required. Algorithm 3.2 would produce only regular vines at the cost of greater programming effort and a larger number of arithmetic operations.

Table 1 shows the number of labeled trees, labeled vines, labeled regular vines and tree-equivalent regular vines, on up to 8 nodes. Table 2 presents the number of labeled regular vines and labeled trees according to unlabeled trees on at most 7 nodes. The number of ways of extending a vine on $n-1$ nodes to an vine on n nodes has been found and the number of labeled regular vines as a function of n has been presented. Future research about efficient implementation and storing of codes for regular vines is desirable.

References

[1] K. Aas, C. Czado, A. Frigessi, and H Bakken. Pair-copula constructions of multiple dependence. *To appear in Insurance: Mathematics and Economics*, 44(1), 2007.

[2] K. Aas and D. Berg. Models for construction of multivariate dependence. *Accepted for publication in European Journal of Finance*, 2009.

[3] A. Min and C. Czado. Bayesian inference for multivariate copulas using pair copula constructions. *Submitted for publication*, 2008.

[4] O. Kolbjornsen and M. Stien. 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), March 2008.

[6] Steven B. Smith. *The Great Mental Calculators. The Psycology, Methods, and Lives of Calculating Prodigies, Past and Present.* Columbia University Press, 1983.

[7] J. Fauvel and P. Gerdes. African slave and calculating prodigy: Bicentenary of the death of thomas fuller. *Historia Mathematica*, (17):141–151, 1990.

[8] F. Harary and E.M. Palmer. *Graphical Enumeration.* Academic Press, 1973.

[9] R.M. Cooke. Markov and entropy properties of tree and vine-dependent variables. In *Proceedings of the ASA Section on Bayesian Statistical Science,*, 1997.

[10] D. Kurowicka and R.M. Cooke. Completion problem with partial correlation vines. *Linear Algebra and its Applications*, 418(1):188 – 200, 2006.

[11] D. Kurowicka and R.M. Cooke. *Uncertainty Analysis with High Dimensional Dependence Modelling.* Wiley, 2006.

[12] A Cayley. A theorem on trees. *The Quarterly Journal of Pure and Applied Mathematics*, 23:376–378, 1889.

[13] J.W. Moon. Various proofs of cayley's formula for counting trees. In Frank Harary, editor, *A Seminar on Graph Theory*, pages 70–78, 1967.

[14] Von H. Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys.*, (27):742–744, 1918.

[15] T.J. Bedford and R.M. Cooke. Vines - a new graphical model for dependent random variables. *Ann. of Stat.*, 30(4):1031–1068, 2002.

[16] L.W. 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.

[17] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[18] F. Harary. Some theorems and concepts of graph theory. In Frank Harary, editor, *A Seminar on Graph Theory*, pages 70–78, 1967.

[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.C. Read and R.E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5:678–692, 1975.

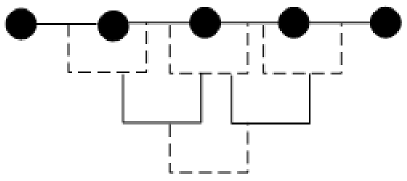
[22] M.J. Smith. Generating spanning trees. Master's thesis, University of Victoria, 1997.

[23] A. Shioura, A. Tamura, and T. Uno. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5, 1975.

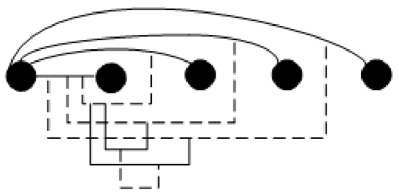
[24] E.R. Sheinerman. Matgraph: A toolbox for graph theory, 2009.

[25] V.N. Kasyanov and V.A. Evstigneev. *Graph Theory for Programmers-Algorithms for Processing Trees.* Kluwer Academic Publishers, 2000.

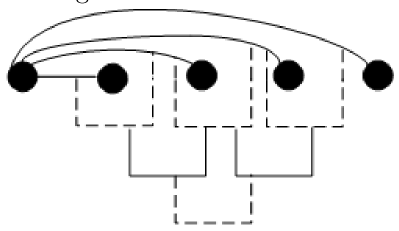
Appendix 1



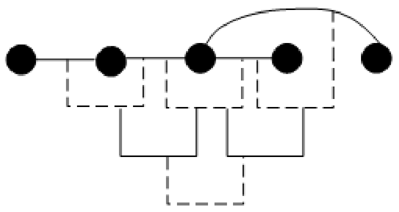
D-vine: Each node in T_i
has degree at most 2.



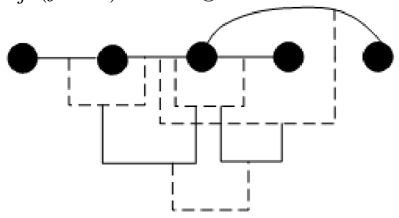
C-Vine: Each tree T_i
has a unique node of degree $n - i$.



Regular Vine where one node in T_1
has degree 4 and each node in
T_j $(j > 1)$ has degree at most 2.



Regular Vine where one node in T_1
has degree 3 and each node in
T_j $(j > 1)$ has degree at most 2.



Regular Vine where one node in T_1
has degree 3, one node in tree T_2
has degree 3 and each node in
T_j $(j > 2)$ has degree at most 2.

Figure A.1: Tree-equivalent regular vines with up to 5 nodes in T_1.