Number of Directed Acyclic Graphs with Extra Constraints

by

Max Schuurman

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Student number: 5369487
Project duration: April 22, 2025 – Jun 26, 2025
Thesis committee: Dr. D. Kurowicka, TU Delft, supervisor
Dr. A.F.F. Derumigny, TU Delft, examinator

TU Delft

Lay Summary

This thesis examines how to count the number of ways in which directed networks without loops are created, known as Directed Acyclic Graphs (DAGs), under various constraints. These graphs are important for modeling tasks, processes, and relationships in which certain steps must precede others, such as scheduling tasks or determining cause-effect relationships. This report adapts a powerful mathematical method originally developed by the mathematician R.W. Robinson, to systematically count these DAGs, even when they must meet specific conditions like having a fixed number of connections or special substructures. This approach gives interesting insights into otherwise complex counting problems, giving a precise count of possible configurations for different networks. The findings presented here help clarify how quickly the complexity of such networks grows as they expand and the techniques and results in this thesis lay the groundwork for tackling even more complicated network structures in the future.

Abstract

This report investigates the enumeration of labeled directed acyclic graphs (DAGs) under various structural constraints, extending an inclusion–exclusion recurrence introduced by R.W. Robinson. Starting from the enumeration of general DAGs via out-point partitioning, the recursive method is adapted to count more specialized classes such as DAGs with a fixed number of arcs or out-points. The Robinson method is also applied to create a formula for the enumeration of rooted directed trees, polytrees, and a special triangle structure. For each of these constrained graph classes, both the closed-form expressions and the derived recursive enumeration formulas are explained, showcasing the versatility and mathematical elegance of Robinson's technique. A main focus of this report is the derivation and interpretation of the Robinson recurrence that adjusts the choose-attach model while using the local attachment rules. This report also presents visual illustrations of the original Robinson method and its adaptations. The results of these formulas are shown in graphs and tables to show the exponential growth of each class. The results bring together several enumeration problems in graph theory under a single recursive framework, offering both theoretical insight and practical enumeration formulas. The contributions of this thesis provide an analysis of DAG enumeration problems, bridging theoretical insights and practical relevance. These results have important implications for a wide range of applications, including Bayesian networks, causal inference modeling, scheduling, and network design. Finally, promising directions for future research are mentioned, emphasizing potential extensions to more complex graph structures and advanced enumeration methodologies.

Contents

1 Introduction

Motivation and Relevance of Labeled Directed Acyclic Graphs

Directed acyclic graphs (DAGs) are essential tools for modeling systems in which directionality and dependency are fundamental. When vertices are labeled, each node explicitly represents a specific entity, and each distinct graph configuration encodes different dependencies or causal relationships. Labeled DAGs commonly represent execution sequences in task scheduling problems, where tasks begin only after prerequisite tasks are completed [12]. Each node corresponds to a distinct task, and the acyclic structure ensures no circular dependencies occur. Figure 1.1 illustrates such a labeled DAG for task scheduling, highlighting how the explicit labeling of tasks (Design, Code, Write Docs, Test, Deploy) clarifies the execution sequence.

DAGs are also fundamental in causal inference, especially within Bayesian networks. In this context, each node represents a specific random variable and the directed edges encode causal relationships [6]. The directionality of edges indicates causality, with each configuration potentially representing a distinct causal system. Figure 1.2 illustrates a labeled DAG modeling causal relationships among variables such as Exercise, Weight, and Heart Health.

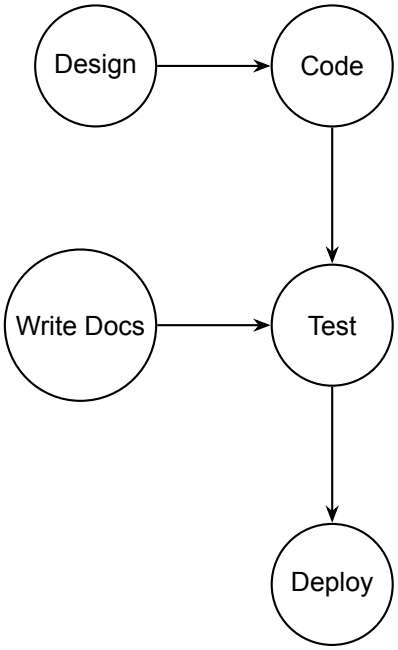


Figure 1.1: Labeled DAG for task scheduling. Explicit labels define task identities and clarify execution sequences without circular dependencies.

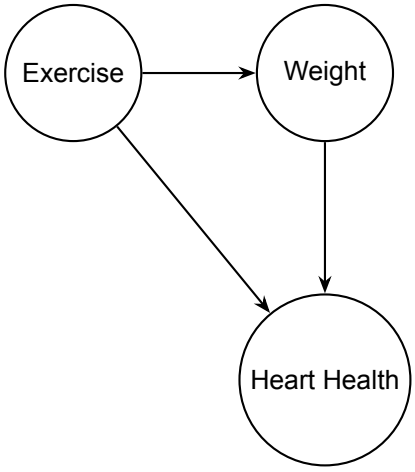


Figure 1.2: Labeled DAG for a causal model. Explicit labeling identifies distinct variables and their directional causal influences.

Enumerating labeled DAGs is essential both theoretically and practically. Determining how many labeled DAGs exist provides crucial insight into the complexity and feasibility of systematic analysis methods, including systematic search, randomized generation, and recursive computation. Because the edge direction encodes dependencies or causal relations, each labeled DAG configuration can represent a distinct scenario. For instance, ”A causes B” differs from ”B causes A,” illustrating the significance of labeling. Therefore, accurate enumeration of labeled DAGs is critical for analyzing, searching, and learning complex systems [5].

The Enumeration Problem

Given p labeled vertices, the enumeration problem asks how many distinct DAGs can be formed among these vertices. The first challenge arises from the global acyclicity constraint, which prohibits cycles and cannot be verified solely by local examination.

One naïve enumeration method is generating all directed graphs on these vertices and filtering out graphs containing cycles. With p labeled vertices, there are $p(p-1)$ possible directed edges, each independently present or absent. Consequently, the number of distinct labeled directed graphs grows exponentially [13]. As p grows, most generated graphs will contain cycles, making it computationally infeasible to check each graph individually for cycles. In fact, results from random graph theory indicate that large random directed graphs almost always contain cycles [9], confirming the impracticality of exhaustive enumeration.

To overcome these computational difficulties, R. W. Robinson proposed a recursive enumeration method that takes advantage of the existence of vertices with zero in-degree (out-points). Robinson’s method recursively decomposes DAG enumeration into smaller subproblems, significantly reducing computational complexity by avoiding direct cycle checking. The approach applies the inclusion-exclusion principle to establish a recurrence relation expressing the count of labeled DAGs on p vertices in terms of DAG counts with fewer vertices. The second part of this report details Robinson’s method and presents its derivation.

Project Objectives and Extensions

After deriving and examining Robinson’s recursive formula, this report explores enumeration extensions by imposing additional constraints. Initially, existing methods for enumerating DAGs with exactly m arcs or exactly k out-points will be reviewed. Enumerating DAGs with a fixed number of arcs significantly narrows the solution space and proves valuable in applications requiring bounded connectivity, such as network design or efficient workflow management. Similarly, constraining the enumeration to exactly k out-points provides clarity and practical relevance for scenarios where specific nodes act explicitly as initiators or source tasks.

Subsequently, this report will apply Robinson’s recursive method to enumerate special classes of DAGs, including rooted directed trees, polytrees, and DAGs composed of triangles. These applications illustrate the versatility of Robinson’s approach and its effectiveness in tackling combinational enumeration problems within specific structural constraints.

2

Graph Theory Definitions

This chapter introduces and elaborates on key graph theory concepts essential for understanding the enumeration of Directed Acyclic Graphs (DAGs). These definitions form the foundational language and tools used in this thesis, these definitions and notations are from [3, 14].

Basic Concepts

Vertex and Edge

A vertex is a fundamental unit in a graph, representing an entity such as a task, event, or data point. An edge is a connection between two vertices.

Graph

A graph is a mathematical structure used to model relationships between pairs of objects. Graphs are widely used in fields such as computer science, logistics, biology, and network analysis.

Definition 1 (Graph)**.** *A graph G is a pair of sets (V, E) where V is non-empty and E is a subset of the set $\{\{u, v\} : u, v \in V, u \neq v\}$ of all two-element subsets of V. The set V is known as the set of **vertices** and the set E is called the set of **edges**.*

Undirected Graph

Definition 2 (Undirected Graph)**.** *An* undirected graph *is a pair $G = (V, E)$ with non-empty vertex set V and edge set $E \subseteq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$ of unordered pairs. Since each edge is unordered, $\{u, v\} \in E \iff \{v, u\} \in E$.*

An undirected graph contains edges without inherent direction: if $\{u, v\} \in E$ then u and v are adjacent in both ways. Such graphs model symmetric relationships, for instance, friendship networks or undirected communication links. An example of an undirected graph is shown in figure 2.1.

Neighbor

Definition 3 (Neighbor)**.** *Vertices u and v are* neighbors *if $\{u, v\} \in E$. The set of all neighbors of a vertex $v \in V$ is denoted by $\mathrm{N}(v) = \{u \in V \mid (\{v, u\} \in E\}$.*

Degree

Definition 4 (Degree)**.** *Let $G = (V, E)$ be an undirected graph. The* degree *of a vertex $v \in V$, denoted $\deg(v)$, is the number of neighbors of v,*

$$\deg(v) = |\{\{u, v\} \in E \mid u \in V \setminus \{v\}\}|.$$

Directed Graph (Digraph)

Definition 5 (Directed Graph)**.** *A* directed graph *(or* digraph*) is a pair $G = (V, E)$, where V is a non-empty set of vertices and $E \subseteq \{(u, v) \mid u, v \in V,\ u \neq v\}$ is a set of ordered pairs of distinct vertices, called* arcs*.*

Since arcs are ordered, the presence of $(u, v) \in E$ *excludes* $(v, u) \in E$*; each direction is treated independently.*

Directed edges (u, v) encode a relationship from u to v. An example of a directed graph is shown in figure 2.1.

Skeleton (Underlying Graph)

Definition 6 (Skeleton (Underlying Graph))**.** *Let* $G = (V, E)$ *be a graph, possibly directed and/or containing multiple edges or loops. The* skeleton *(or* underlying graph*) of* G*, denoted* $\mathrm{skel}(G)$*, is the undirected graph* (V, E')*, where* $\mathrm{skel}(G)$ *retains the same vertex set as* G*, with a single undirected edge between any two adjacent vertices, ignoring direction, multiplicity, and loops,*

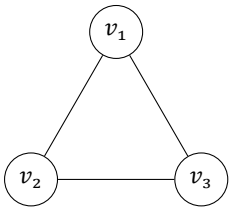
$$E' = \big\{\{u, v\} \mid u \neq v \text{ and } G \text{ contains an edge or arc between } u \text{ and } v\big\}.$$

Cycle

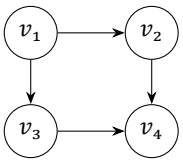
A cycle in an undirected graph is a closed path where vertices are revisited only at the start/end, and edges can be traversed in either direction. In a directed graph, a cycle requires all arcs to follow a consistent direction. Directed cycles are fundamental in detecting feedback loops or violations of acyclicity.

Definition 7 (Cycle)**.** *Let* $G = (V, E)$ *be a graph.*

- *In an* undirected graph*, a* cycle *is a sequence of distinct vertices* $v_1, v_2, \ldots, v_k \in V$ *with* $k \geq 3$*, such that* $\{v_i, v_{i+1}\} \in E$ *for all* $1 \leq i < k$*, and* $\{v_k, v_1\} \in E$*. All vertices are distinct except that* $v_1 = v_{k+1}$*, completing the loop.*
- *In a* directed graph *(digraph), a* cycle *is a sequence of distinct vertices* $v_1, v_2, \ldots, v_k \in V$ *with* $k \geq 2$*, such that* $(v_i, v_{i+1}) \in E$ *for all* $1 \leq i < k$*, and* $(v_k, v_1) \in E$*. The direction of the arcs must follow the sequence strictly, and again, all vertices are distinct except* $v_1 = v_{k+1}$*.*



(a) Undirected graph on 3 nodes (Undirected triangle)



(b) Directed acyclic graph on 4 nodes. The underlying graph has a cycle of length 4.

Figure 2.1: (a) An undirected graph containing a cycle; (b) a directed acyclic graph.

Tail and Head (Parent and Child)

In a directed graph, the direction of each arc matters. The tail (parent) is the vertex where the arc originates, and the head (child) is where it terminates.

Definition 8 (Tail and Head (Parent and Child))**.** *Let* $G = (V, E)$ *be a directed graph and let* $(u, v) \in E$ *be an arc. The vertex* u *is called the tail of the arc or parent of the node* v*, and* v *is called the head of the arc or a child of u.*

That is, the arc (u, v) *is directed from* u *to* v*, indicating a one-way relationship from tail to head.*

Orientation

An orientation of an undirected graph turns it into a directed graph by choosing a direction for every edge. This process preserves the connectivity structure while introducing directionality.

Definition 9 (Orientation)**.** *Let* $G = (V, E)$ *be an undirected graph, where* $E \subseteq \{\{u, v\} \mid u, v \in V,\ u \neq v\}$*. An* orientation *of* G *is a directed graph* $\vec{G} = (V, \vec{E})$*, where* $\vec{E} \subseteq \{(u, v) \mid \{u, v\} \in E\} \cup \{(v, u) \mid \{u, v\} \in E\}$*, such that for every edge* $\{u, v\} \in E$*, exactly one of* (u, v) *or* (v, u) *belongs to* $\vec{E}$*.*

Topological Order

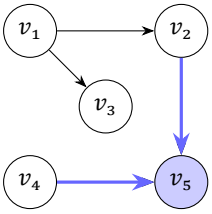
Definition 10 (Topological Order)**.** *A* topological order *of a directed graph $G = (V, E)$ is a total order on V such that for every arc $(u, v) \in E$, it holds that $u < v$. Equivalently, it is a labeling of the vertices with distinct integers $1, 2, \ldots, |V|$ such that all arcs point from smaller to larger labels.*

In-Degree and Out-Degree

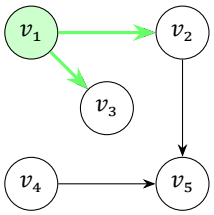
In a directed graph, the in-degree of a vertex counts how many edges point *into* it, and the out-degree counts how many edges *leave* it.

Definition 11 (In-Degree and Out-Degree)**.** *Let $G = (V, E)$ be a directed graph. For a vertex $v \in V$, the* in-degree *of v, denoted $\deg^-(v)$, is the number of arcs $(u, v) \in E$ with head v. The* out-degree *of v, denoted $\deg^+(v)$, is the number of arcs $(v, u) \in E$ with tail v. Formally,*

$$\deg^-(v) = |\{u \in V \mid (u, v) \in E\}|, \quad \deg^+(v) = |\{u \in V \mid (v, u) \in E\}|.$$



(a) Vertex v_5 has $\deg^-(v_5) = 2$.



(b) Vertex v_1: $\deg^+(v_1) = 2$ and $\deg^-(v_1) = 0$.

Figure 2.2: Illustrating vertex properties on the same DAG.

Out-point

Definition 12 (Out-point)**.** *In a directed graph, an* out-point *is a vertex $v \in V$ with in-degree zero: $\deg^-(v) = 0$.*

Such vertices typically represent starting points in a process or independent tasks. In figure 2.2, vertex v_1 is the only out-point, serving as the root of the dependency structure.

Path

A path represents a walk through the graph, traversing edges (or arcs) from one vertex to another. In directed graphs, the direction of arcs must be followed, whereas in undirected graphs, edges can be traversed in either direction. Paths are central to many graph algorithms and structural properties.

Definition 13 (Path)**.** *Let $G = (V, E)$ be a graph.*

- *In an* undirected graph*, a* path *is a sequence of vertices $v_1, v_2, \ldots, v_k \in V$ such that $\{v_i, v_{i+1}\} \in E$ for all $1 \leq i < k$. The path is* simple *if all vertices are distinct.*
- *In a* directed graph *(digraph), a* path *is a sequence of vertices $v_1, v_2, \ldots, v_k \in V$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$. The path is* simple *if all vertices are distinct.*

Types of Graphs

Labeled Graph

Definition 14 (Labeled Graph)**.** *A* labeled graph *is a graph where each vertex $v \in V$ has a unique identifier, usually from the set $\{1, 2, \ldots, p\}$. Two labeled graphs are equal only if they have the same structure and the same vertex labels.*

Unlabeled Graph

Definition 15 (Unlabeled Graph)**.** *An* unlabeled graph *is considered up to isomorphism; only the structure (adjacency relationships) matters, not the labels of the vertices. Two unlabeled graphs are considered the same if there exists a bijection between their vertex sets that preserves adjacency.*

Directed Acyclic Graph (DAG)

Definition 16 (Directed Acyclic Graph)**.** *A* Directed Acyclic Graph (DAG) *is a directed graph* $G = (V, A)$ *with no directed cycles; that is, there do not exist distinct vertices* $v_1, \ldots, v_k$ *such that* $(v_i, v_{i+1}) \in A$ *for* $1 \leq i < k$ *and* $(v_k, v_1) \in A$.

In Figure 2.1(b) a DAG on 4 nodes is shown. There is no path in this graph that starts and ends at the same node.

Tree

A tree is a minimal connected undirected graph: removing any edge disconnects the graph. It is also a maximal acyclic graph: adding any new edge creates a cycle. Trees are fundamental in hierarchical structures and recursive algorithms.

Definition 17 (Tree)**.** *An* undirected tree *is a connected undirected graph* $G = (V, E)$ *with no cycles. That is,* T *is connected and acyclic.*

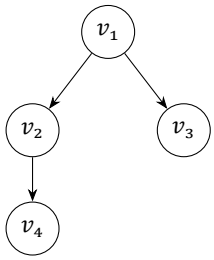
Equivalently, a tree is an undirected graph in which any two vertices are connected by exactly one simple path.

Rooted Directed Tree

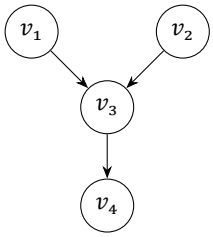
Definition 18 (Rooted Directed Tree)**.** *A* rooted directed tree *is a directed graph* $G = (V, E)$ *with a distinguished root vertex* $r \in V$ *such that the underlying undirected graph is a tree, the root* r *has indegree 0, every other vertex* $v \in V \setminus \{r\}$ *has indegree 1, and there exists a unique directed path from each vertex* $v \in V \setminus \{r\}$ *to the root* r.

Polytree

Definition 19 (Polytree)**.** *A* polytree *is a directed acyclic graph* $G = (V, E)$ *whose underlying undirected graph is a tree; that is, it is connected and contains no undirected cycles.*



(a) Rooted directed tree



(b) Polytree

Figure 2.3: (a) A rooted directed tree; (b) a polytree where v_3 has two parents.

Bipartite Graph

Bipartite graphs model relationships between two distinct classes of objects, for example, students and courses, jobs and machines, or users and movies.

Definition 20 (Bipartite Graph)**.** *A graph* $G = (V, E)$ *is bipartite if there exists a partition of the vertex set into two disjoint subsets* V_1 *and* V_2 *(called a bipartition) such that every edge joins one vertex of* V_1 *to one vertex of* V_2*:*

$$E \subseteq \{\{u, v\} \mid u \in V_1,\ v \in V_2\}.$$

Equivalently, a graph is bipartite if and only if it contains no cycle of odd length.

3

Robinson's Recursive Formula

This chapter presents a recursive formula for enumerating labeled DAGs, originally derived by R. W. Robinson [10]. Robinson's recursive method allows for a combinatorial interpretation that offers deeper insight into the structure of DAGs. The focus of this section is to provide a detailed walk-through of this derivation from Robinson. This will serve as the foundation for later sections, where further extensions of the formula are explored.

Breakdown of Robinson's Recursive Formula

Robinson introduced a recursive formula based on the principle of inclusion-exclusion and the concept of out-points. The recursive formula is given by:

Robinson Let a_p denote the number of labeled DAGs on p vertices. Then

$$a_p = \sum_{s=1}^{p} (-1)^{s+1} \binom{p}{s} 2^{s(p-s)} a_{p-s},$$

with the initial condition $a_0 = 1$. where:

- a_p is the number of labeled DAGs on p vertices,
- s is the number of vertices that are out-points (vertices with in-degree zero) in a DAG,
- $\binom{p}{s}$ is the number of ways to choose s out-points from the p labeled vertices,
- $2^{s(p-s)}$ counts all possible directed edges from the s out-points to the remaining $p - s$ vertices,
- a_{p-s} is the number of labeled DAGs on the remaining $p - s$ vertices.

Step by Step Derivation

A key observation for the derivation of this formula is that every nonempty DAG must contain at least one out-point that is, a vertex with in-degree zero. This can be understood by contradiction: suppose that a nonempty DAG has no out-point. Then every vertex has at least one incoming edge, which implies the existence of a closed directed path. However, this would form a directed cycle, which contradicts the definition of a DAG. Therefore, at least one out-point must exist in every DAG.

To count all labeled DAGs, Robinson grouped all possible DAGs according to their sets of out-points. However, since a single DAG with multiple out-points belongs to multiple such groups, one for each subset of its out-points, this leads to overcounting. To correct for this, the principle of inclusion-exclusion is applied: overlapping contributions from subsets are alternately added and subtracted to ensure that each DAG is counted exactly once. This combinatorial insight leads to an elegant recursive formula for enumerating labeled DAGs.

Step 1: at least one out-point

Figure 3.1 shows a DAG on five vertices and highlights its unique out-point. The existence of such a vertex motivates the entire decomposition.

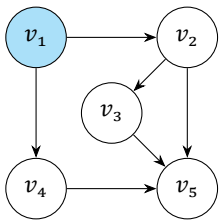


Figure 3.1: A DAG on $p = 5$ vertices. Vertex v_1 (blue) is an out-point because no edge enters it.

Step 2: choose a set S of out-points

Let $S \subseteq \{1, 2, \ldots, p\}$ be a chosen set of s vertices that will be out-points. First, the presence or absence of each of the $s(p-s)$ possible arcs from the subset S to the remaining $p-s$ vertices must be determined. Since each arc can independently be present or absent, there are $2^{s(p-s)}$ possible configurations. Once those outward arcs are fixed, the remaining $p - s$ vertices must themselves form any labeled DAG, of which there are a_{p-s}. Together, the number of possible labeled DAGs on p vertices with these s outpoints is $2^{s(p-s)}\, a_{p-s}$, or formally

$$\Big|\bigcap_{i \in S} P_i\Big| = 2^{s(p-s)}\, a_{p-s}.$$

Figure 3.2 illustrates these free choices for $S = \{v_1, v_2\}$.

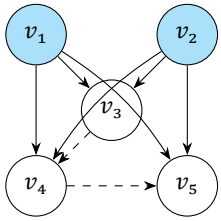


Figure 3.2: Fixing $S = \{v_1, v_2\}$ ($s = 2$). The 6 solid arrows represent the $s(p-s)$ outward edge possibilities. Each may be present or absent, giving 2^6 options. The dashed arrows show an example of a DAG on the remaining vertices.

Interpretation of Each Term

Combining the preceding components gives the following recursive formula:

$$\binom{p}{s} 2^{s(p-s)}\, a_{p-s}.$$

The three factors have a transparent meaning:

- **Choosing S.** There are $\binom{p}{s}$ ways to choose the out-point set.
- **Edge freedom.** For every ordered pair (u, v) with $u \in S$ and $v \in V \setminus S$ the arc $u \to v$ can be included or excluded. There are $s(p-s)$ such pairs, giving $2^{s(p-s)}$ configurations.
- **Internal structure.** After these choices, the vertices in $V \setminus S$ must form a labeled DAG, of which there are a_{p-s}.

The Role of Inclusion–Exclusion

If all of the above values are simply summed, the resulting formula would overcount the number of DAGs. A graph with, for example, three out-points is included once for each non-empty subset of

those out-points. Inclusion–exclusion solves this problem by adding the $s = 1$ counts, subtracting the $s = 2$ counts, adding the $s = 3$ counts, and so on, exactly the following alternating pattern

$$0 = \sum_{\emptyset \neq S \subseteq [p]} (-1)^{|S|+1} \Big| \bigcap_{i \in S} P_i \Big|.$$

Figure 3.3 shows the $s = 2$ case of the alternation.

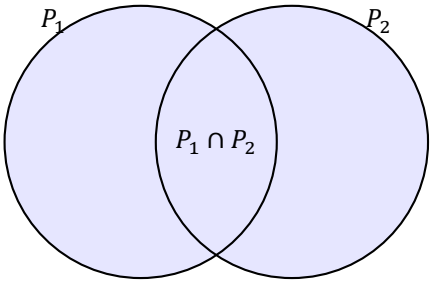


Figure 3.3: A two-set Venn diagram: the overlap is added in with $s = 2$ and subtracted out again when $s = 1$. The alternating signs guarantee every DAG is counted exactly once.

Setting $s = |S|$ this yields Robinson's recurrence

$$a_p = \sum_{s=1}^{p} (-1)^{s+1} \binom{p}{s} 2^{s(p-s)} a_{p-s}, \qquad a_0 = 1.$$

Advantages of the Recursive Approach

Robinson's recursive formula offers a systematic and scalable method to enumerate labeled DAGs. By decomposing the counting problem into smaller subproblems based on the number of out-points, the recurrence enables efficient computation for moderate values of p without the need to generate all possible graphs explicitly. In addition to its computational utility, the formula also reveals structural insights in the set of DAGs, framed in terms of their out-point configurations. This will help to extend the approach to more refined enumeration problems involving additional constraints.

Exclusion of Cyclic Graphs in the Robinson Method

The recursive method developed by Robinson for enumerating labeled DAGs excludes graphs containing cycles. This follows from a key structural property of DAGs: every nonempty DAG must contain at least one vertex with in-degree zero, known as an out-point.

In the Robinson approach, the enumeration is carried out by selecting subsets $S \subseteq V$ of out-points, constructing all valid edge configurations from S to $V \setminus S$, and recursively counting DAGs on the remaining vertices. Crucially, graphs containing cycles cannot satisfy the out-point condition, as every vertex in a directed cycle has positive in-degree. Such graphs are therefore excluded from all intersections $\bigcap_{i \in S} P_i$ considered in the inclusion-exclusion formula.

Furthermore, the recursive construction only combines smaller acyclic graphs with edge sets directed from out-points to the rest of the graph, preserving acyclicity at each stage.

Theorem 3.0.1 (Robinson)**.** *Let a_p denote the number of labeled DAGs on p vertices. Then*

$$a_p = \sum_{s=1}^{p} (-1)^{s+1} \binom{p}{s} 2^{s(p-s)} a_{p-s}, \tag{3.1}$$

with the initial condition $a_0 = 1$.

Proof. Let $V = \{1, 2, \ldots, p\}$ denote the set of labeled vertices. Define, for each $i \in V$, the set P_i as the collection of all labeled DAGs on V in which the vertex i is an out-point, which means that vertex i has in-degree zero.

Since every nonempty DAG must contain at least one out-point, it follows that:

$$a_p = \left| \bigcup_{i \in V} P_i \right|.$$

Where $\bigcup_{i \in V} P_i$ represents the union of all such sets P_i, that is, the set of all DAGs in which at least one vertex is an out-point. The outer vertical bars $|\cdot|$ denote the size of the set, that is, the number of distinct DAGs captured by at least one P_i.

However, the sets P_i are generally not disjoint: a single DAG with multiple out-points can belong to several of the sets P_i. To accurately count the total number of labeled DAGs without overcounting those that appear in multiple sets, the principle of inclusion-exclusion is applied. This principle corrects for overlapping elements by alternately adding and subtracting the sizes of intersections of the sets P_i. This yields the following formula:

$$\left| \bigcup_{i \in V} P_i \right| = \sum_{\emptyset \neq S \subseteq V} (-1)^{|S|+1} \left| \bigcap_{i \in S} P_i \right|.$$

In this expression, the summation symbol $\sum_{\emptyset \neq S \subseteq V}$ denotes a sum taken over all non-empty subsets S of the vertex set V. For each such subset S, the intersection $\bigcap_{i \in S} P_i$ denotes the set of all DAGs in which every vertex in S has in-degree zero. The alternating sign $(-1)^{|S|+1}$ ensures that overcounted graphs are adjusted correctly.

Fix a non-empty subset $S \subseteq V$ with $|S| = s$. The intersection $\bigcap_{i \in S} P_i$ consists of all labeled DAGs in which each vertex in S is an out-point. In such graphs:

- No incoming edges are allowed into any vertex of S,
- Directed edges from vertices in S to vertices in $V \setminus S$ can either be included or excluded independently,
- The induced subgraph on $V \setminus S$ must itself be a labeled DAG.

There are $s(p-s)$ possible directed edges from the vertices in S to the vertices in $V \setminus S$. Since each edge can either be present or absent, there are $2^{s(p-s)}$ possible configurations of these edges. The number of labeled DAGs that can be formed on the remaining $p-s$ vertices is, by definition, a_{p-s}. Therefore, the total number of labeled DAGs corresponding to a fixed set S of out-points is:

$$\left| \bigcap_{i \in S} P_i \right| = 2^{s(p-s)} a_{p-s}.$$

Since there are $\binom{p}{s}$ ways to choose a subset S of size s, the total contribution from all subsets of size s is:

$$\binom{p}{s} 2^{s(p-s)} a_{p-s}.$$

By the inclusion-exclusion principle, this term had to be multiplied by $(-1)^{s+1}$ to account for overcounting subsets of different sizes.

Summing over all $s = 1, \ldots, p$, Robinson's recursive formula is obtained:

$$a_p = \sum_{s=1}^{p} (-1)^{s+1} \binom{p}{s} 2^{s(p-s)} a_{p-s}.$$

Finally, the initial condition $a_0 = 1$ states that there is exactly one DAG (the empty graph) on zero vertices, completing the proof. □

Visualizing All Labeled DAGs on Three Nodes

To provide the reader with a concrete understanding of the enumeration problem, Figure 3.4 presents all possible labeled DAGs on three nodes. The DAGs are sorted by the number of directed edges (arcs), ranging from 0 to 3.

Each DAG is drawn with labeled vertices and a unique acyclic orientation of the edges. The sorting by edge count helps illustrate how the number of valid DAGs grows rapidly as edges are added, but also how the acyclicity constraint prevents the formation of certain edge combinations. This example concretely demonstrates the structural constraints involved in DAG enumeration and motivates the recursive and closed-form methods developed in later sections.

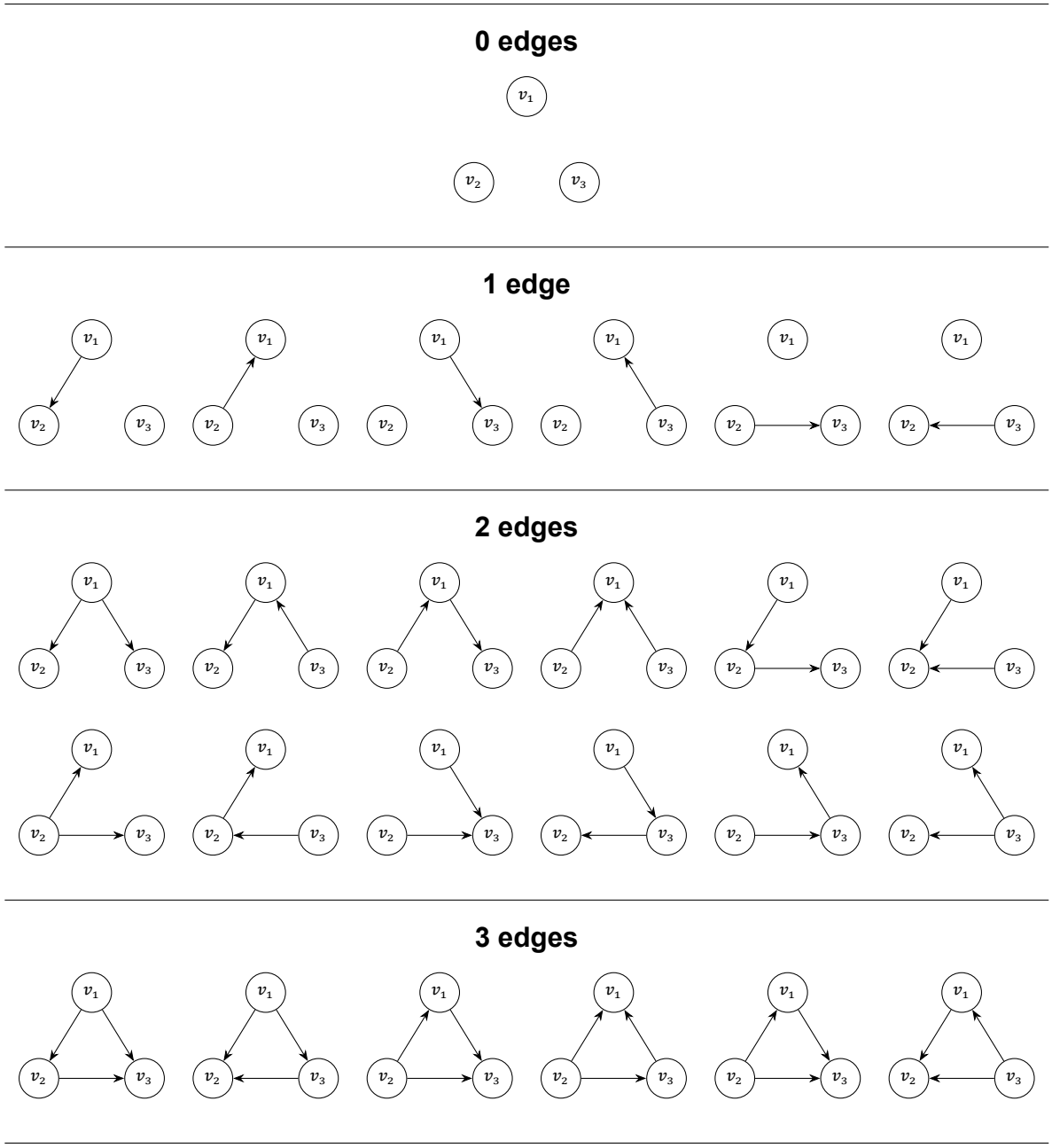


Figure 3.4: All 25 possible labeled DAGs on $\{v_1, v_2, v_3\}$, grouped by edge-count (0, 1, 2, and 3 edges).

4

Enumerating DAgs With Extra Constraints Using Robinson's Argument

In this chapter, Robinson's classical recursive enumeration approach is extended to address more specialized combinational problems involving DAGs. Specifically, the enumeration of labeled DAGs is considered under additional structural constraints, such as a fixed number of arcs or a fixed number of out-points. By adapting Robinson's original inclusion-exclusion framework, refined recursive formulas are developed that efficiently handle these extra conditions. The methods and results presented in this chapter form an essential foundation for further investigation of specialized DAG subclasses and their enumeration.

4.1. Existing Extensions of The Robinson Method

In this section, the existing extensions and refinements of Robinson's classical recursive enumeration technique for labeled DAGs are explored. Although Robinson's original method provided a robust framework for general enumeration, some researchers have expanded upon his foundational approach to address specific combinational challenges. Methods developed for the enumeration of DAGs with constraints such as a fixed number of arcs or a fixed number of out-points are examined. Reviewing these existing methodologies not only highlights the adaptability and versatility of Robinson's approach but also sets the stage for further applications.

4.1.1. Enumeration of Labeled Directed Acyclic Graphs with a Fixed Number of Arcs

The classical result of Robinson [10] gives a neat inclusion-exclusion derivation for the number a_p of labeled acyclic digraphs (DAGs) on p vertices. In many practical applications, especially in probabilistic modeling and network design, it is often of interest to enumerate not only all possible labeled DAGs, but also those with a fixed number of arcs. For example, in workflow scheduling systems, there may be a strict upper limit on task dependencies due to hardware or resource constraints. This section develops a two-parameter recurrence that extends Robinson's argument [11].

Setting up the Parameters

Let $a_{p,m}$ denote the number of labeled DAGs on p vertices with exactly m arcs, formally

$$a_{p,m} = \left|\{\text{labeled DAGs on } p \text{ vertices with exactly } m \text{ arcs}\}\right|,$$

with the conventions $a_{0,0} = 1$ and $a_{p,m} = 0$ whenever $m < 0$ or $m > \frac{p(p-1)}{2}$.

For every vertex i define the event

$$P_i = \{\text{DAGs in which vertex } i \text{ has in–degree } 0\}, \qquad i = 1, \dots, p.$$

Because every non-empty DAG has at least one out-point (a vertex of in–degree 0), the union of all such sets $\bigcup_{i \in V} P_i$ covers the whole set of DAGs, making it suitable for inclusion–exclusion.

Counting an intersection while tracking arcs

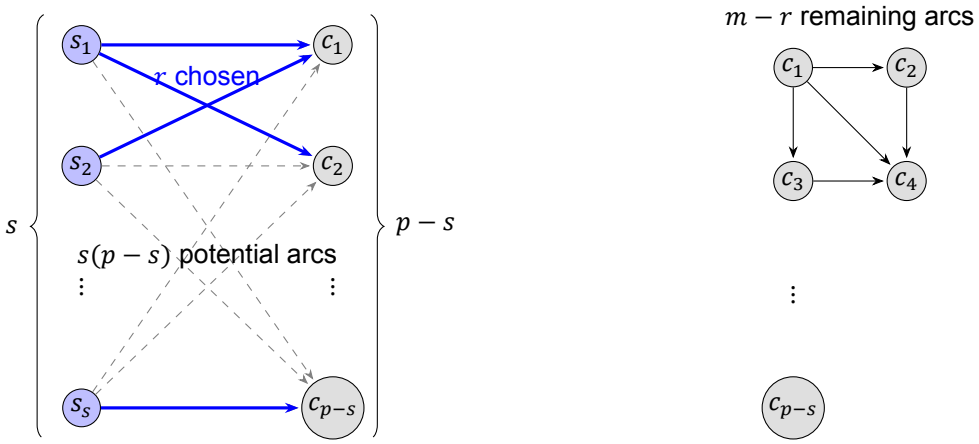
Let $S \subseteq \{1, \dots, p\}$ be a non-empty index set of size s, and define

$$P_S := \bigcap_{i \in S} P_i.$$

A DAG lies in P_S precisely when every vertex in S has in-degree 0. The goal in this step is to count how many DAGs are contained in the fixed intersection P_S and have a prescribed total number of arcs. The following argument gives a step-by-step explanation.

1. Why arcs can only leave S. Because the in-degree of each vertex in S must be zero, no arc may point into S from its complement $\overline{S} = \{1, \dots, p\} \setminus S$, and no arc may exist between elements of S. Hence, all permissible arcs incident to S must originate in S and end in $\overline{S}$.

Therefore, the only possible DAGs are those for which the elements of S are parents of the elements of $\overline{S}$. This structural constraint implies that all valid arcs flow from S to $\overline{S}$, forming a directed bipartite graph. This bipartite configuration is illustrated in the figure 4.1, where $S = \{s_1, \dots, s_s\}$ and $\overline{S} = \{c_1, \dots, c_{p-s}\}$, with all arcs directed from S toward $\overline{S}$.



(a) Step (a): pick r of the $s(p-s)$ potential outgoing arcs.

(b) Step (b): build a DAG on $p-s$ vertices with $m-r$ arcs.

Figure 4.1: Two–stage construction used in $|P_S \cap \{m \text{ arcs}\}|$. First select an *outgoing layer* of r arcs (4.1a); then recursively complete the DAG on the complement with the remaining $m-r$ arcs (4.1b).

2. Counting the outgoing layer. The pair $(S, \overline{S})$ gives rise to exactly $s(p-s)$ possible arcs, since every vertex of S can have an arc to each of the $p-s$ vertices outside S. Let r denote the chosen number of such arcs. Two elementary but crucial bounds hold:

1. $0 \le r \le s(p-s)$ by definition of a subset;
2. $r \le m$ because the entire DAG may contain at most m arcs.

Hence

$$0 \le r \le \min\{m, s(p-s)\}.$$

Given r, the outgoing layer can be selected in $\binom{s(p-s)}{r}$ ways.

3. Recursive completion of the complement. Once the outgoing layer corresponding to S has been fixed, the attention shifts to the remaining vertex set of size $p-s$. Importantly, since no arcs are directed into S, this separation preserves the acyclicity of the subgraph induced by the complement. Consequently, the task of completing the construction reduces to selecting

$$m - r$$

arcs that form a DAG on exactly $p-s$ labeled vertices. By the definition of the double-indexed sequence, there are $a_{p-s,m-r}$ possible configurations.

4. Combining steps (1)–(3). The independence between the choice of the outgoing layer and the internal arrangement of $\overline{S}$ multiplies the possibilities:

$$|P_S \cap \{m \text{ arcs}\}| = \sum_{r=0}^{\min\{m, s(p-s)\}} \binom{s(p-s)}{r} a_{p-s,\,m-r}$$

This equation is the heart of the derivation: it quantifies exactly how many arc budgets are consumed by fixing the out–points S.

Inclusion–exclusion

Applying inclusion-exclusion over all s–element subsets S results in the desired two–parameter recurrence.

Theorem 4.1.1 (Two-Parameter Inclusion–Exclusion Recurrence for Labelled DAGs)**.** *Let $a_{p,m}$ denote the number of labeled DAGs on p vertices with exactly m arcs. Then, for all $p \geq 1$,*

$$a_{p,m} = \sum_{s=1}^{p} (-1)^{s+1} \binom{p}{s} \sum_{r=0}^{\min\{m, s(p-s)\}} \binom{s(p-s)}{r} a_{p-s,\,m-r}\,. \tag{4.1}$$

with initial condition $a_{0,0} = 1$ and $a_{p,m} = 0$ whenever $m < 0$ or $m > \frac{p(p-1)}{2}$.

For m summed over all possible values, this formula returns to Robinson's original single-parameter recurrence.

4.1.2. Counting Labeled DAGs with Exactly k Out-points

In specific contexts, it is of interest to enumerate labeled DAGs that posses exactly k out-points. For example, in task scheduling and project planning, vertices may represent tasks, and having a fixed number of out-points ensures that a certain number of tasks depend on others without introducing additional complexity. Enumeration of such DAGs is achieved by modifying Robinson's recurrence to account only for configurations in which exactly k vertices are selected as out-points [7].

The derivation proceeds by reversing the process; initially, the k out-points are removed, leaving a smaller DAG with $m = p - k$ vertices, among which s nodes are specified as out-points. The required DAGs are then obtained by introducing k new out-points and allowing all possible connections from these to the existing vertices. Each of the $m - s$ non out-point nodes has 2^k possibilities to have an arc to one of the k new out-points or not. Furthermore, the s old out-points must be connected to at least one of the new outpoints, which introduces a factor of $2^k - 1$ to ensure that all out-points are connected. Finally, the labels of the vertices can be rearranged in $\binom{p}{k}$ ways. Figure 4.2 shows an example of how this process works. This leads to the following recursions for counting labeled DAGs with exactly k out-points.

Theorem 4.1.2. *Let $a_{p,k}$ denote the number of labeled DAGs with p vertices and exactly k out-points, for $1 \leq k \leq p$, and set $m = p - k$. Then*

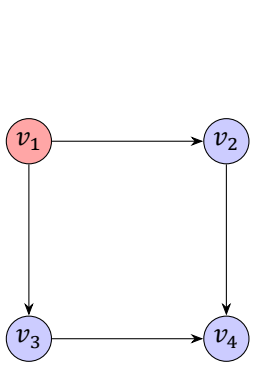
$$a_{p,k} = \binom{p}{k} b_{p,k}, \quad b_{p,k} = \sum_{s=1}^{m} (2^k - 1)^s \, 2^{k(m-s)} \, a_{m,s}. \tag{4.2}$$

with base cases $a_{p,p} = 1$ and $b_{p,p} = 1$.

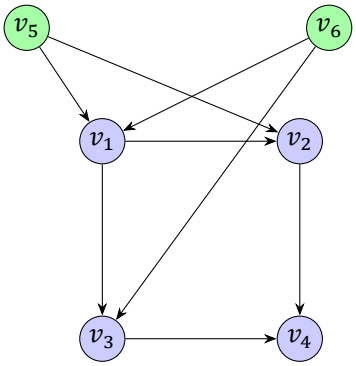
In this formula:

- $\binom{p}{k}$ counts the number of ways to choose k out-points from the p vertices,
- $(2^k - 1)^s$ ensures that all s old out-points are connected to at least one of the new out-points,
- $2^{k(m-s)}$ counts the number of ways to connect the $m - s$ non-outpoint nodes to the k new out-points,
- $a_{m,s}$ counts the number of valid DAGs on the m remaining vertices with exactly s out-points.

This recursive approach enables efficient enumeration of labeled DAGs with exactly k out-points, while systematically ensuring that all structural constraints are satisfied at each step.



(a) Smaller DAG: $m = 4$ vertices with exactly $s = 1$ out-point (v_1, red).

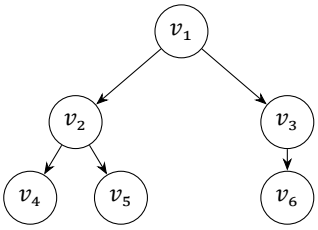


(b) Add $k = 2$ new out-points (v_5, v_6, green). v_1 now has indegree ≥ 1.

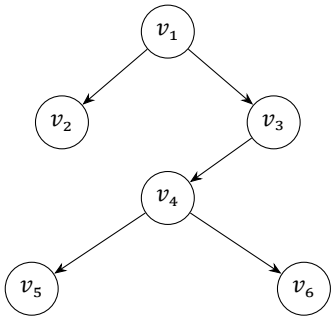
Figure 4.2: Reverse-construction step for counting labeled DAGs with exactly k out-points (sources). Remove the k sources to obtain the smaller graph (left), then add k new sources and all admissible outgoing arcs (right). Each former non-source chooses any subset of incoming arcs from the new sources (2^k options), whereas each former source must pick at least one ($(2^k - 1)$ options), yielding the factors in the recurrence.

4.2. Enumeration of Rooted Directed Trees

Rooted directed trees represent a fundamental class of acyclic directed graphs, where every vertex except the root has a unique parent, and all edges are oriented away from the root. The enumeration of labeled rooted directed trees plays an important role in both combinatorics and computer science. Rooted directed trees arise naturally in a wide range of structures, including data hierarchies, syntax trees, decision processes, and communication networks. This chapter presents both the closed-form formula and the recurrence based on Robinson's argument for counting the number of labeled rooted directed trees, thereby establishing a foundation for subsequent generalizations to other classes of directed acyclic graphs. Figure 4.3 shows two rooted directed trees on six labeled vertices: In (a) the branching factor is balanced between the two children of the root, while in (b) the structure extends in a deeper chain.



(a) Rooted directed tree



(b) Another rooted directed tree

Figure 4.3: Two examples of rooted directed trees.

4.2.1. Closed Form Enumeration of Rooted Directed Trees

In this section, Cayley's classical result will be used to derive the number of possible rooted directed trees, this result also provides the building blocks for enumerating more complex DAG families such as polytrees. In algorithmic contexts, understanding the number of possible tree structures is crucial to analyze the complexity of tree-based searches, optimizations, and storage requirements. In probabilistic modeling, such as Bayesian networks, trees often represent causal structures, and enumeration helps quantify the hypothesis space for structure learning.

Derivation of the Closed Form Formula

Step 1 – Undirected Skeleton. Regardless of the edge directions, each rooted directed tree corresponds to a labeled undirected tree. Conversely, any labeled undirected tree can be transformed into

a rooted directed tree by selecting a root and orienting all edges away from this vertex.

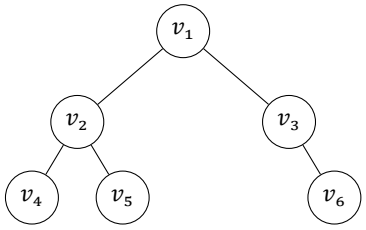
Theorem 4.2.1 (Cayley's Theorem [1])**.** *The number of labeled undirected trees on p vertices is given by*

$$U_p = p^{p-2}. \tag{4.3}$$

This result applies to all labeled undirected trees, which serve as the undirected skeletons for the rooted directed trees considered in this enumeration.

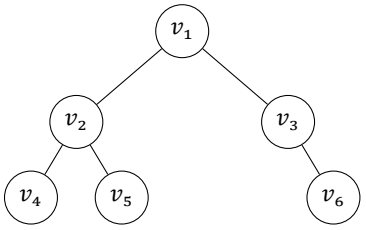
Derivation of Cayley's Formula Using the Prüfer Code Cayley's result can be derived via Prüfer codes, which establish a bijection between labeled undirected trees on p vertices and sequences of length $p-2$ over the set $\{1, 2, \dots, p\}$. The encoding process proceeds as follows:

At each step, the leaf with the smallest label is removed, and its unique neighbor is recorded. Repeating this $p-2$ times produces a sequence known as the Prüfer code. This procedure is reversible: given any sequence of length $p-2$, the unique tree can be reconstructed by maintaining degree counts and repeatedly connecting the smallest available degree-1 vertex to the next symbol in the sequence. The final edge connects the last two remaining vertices. Figure 4.4 shows how the Prüfer code works.



Prüfer code: $(v_2,\ v_2,\ v_1,\ v_3)$

(a) Delete v_4, v_5, v_2, v_1 in that order.



Reconstructed from $(v_2,\ v_2,\ v_1,\ v_3)$

(b) Reconstruction from Prüfer code.

Figure 4.4: Cayley's theorem via Prüfer codes: (a) encoding by successive leaf deletion; (b) decoding by connecting smallest degree-1 vertex to next symbol in the code.

Since there are p choices for each of the $p-2$ entries in the sequence, the total number of labeled undirected trees is:

$$|\text{labeled undirected trees on } p \text{ vertices}| = p^{p-2}.$$

Step 2 – Choose a root. Given a tree structure (the undirected skeleton), any of the p vertices can serve as the root. This multiplies the count by a factor of p:

$$|\text{rooted undirected trees}| = p \cdot p^{p-2}.$$

Step 3 – Orient the tree. Once the root is specified, every edge can be oriented away from the root. This results in a unique directed tree with all arcs pointing outward. This is valid because every vertex (except the root) has a unique parent in a tree, ensuring that each arc has a clear direction.

Closed Form Enumeration of Rooted Directed Trees

Let T_p denote the number of labeled rooted directed trees on the vertex set $V = \{1, 2, \dots, p\}$, where all arcs are oriented away from a single root. As established in the preceding subsection, this enumeration is closely related to Cayley's classical result for undirected trees.

Theorem 4.2.2 (Enumeration of Labeled Rooted Directed Trees)**.** *The number of labeled rooted directed trees on p vertices, where each edge is oriented away from the root, is given by*

$$T_p = p^{p-1}, \qquad \text{for } p \geq 1. \tag{4.4}$$

4.2.2. Enumeration of Rooted Directed Trees Using Robinson's Arguments

Although a closed-form expression already exists for enumerating directed trees, this section derives a recursive formula for counting labeled directed trees on p vertices, each rooted at a unique node. This recursion, inspired by Robinson's inclusion–exclusion techniques for counting acyclic digraphs [10], provides a constructive approach to computing T_p, the number of such rooted trees.

Construction Using a Core and an Attachment Set

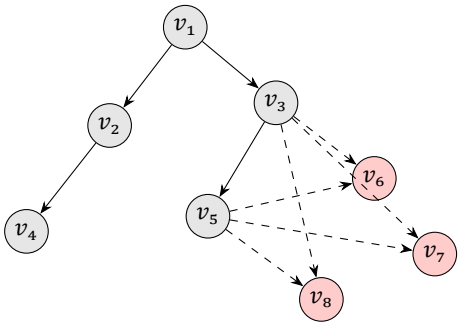
Derivation of the recurrence proceeds by partitioning the vertex set $V = \{1, \dots, p\}$ into two disjoint subsets [2]:

- A *core* C of size $p - k$, which contains the root and forms a connected directed tree. The core represents the portion of the tree that is already fully constructed and to which new vertices can be attached.
- An *attachment set* K of size k, consisting of vertices that must be attached to the core. The attachment set contains those vertices not yet integrated into the tree structure; each such vertex will be connected to a vertex in the core, thus extending the tree while maintaining its rooted and acyclic nature.

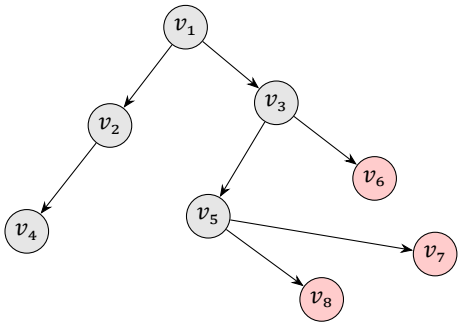
This partitioning enables a recursive construction of larger trees from smaller ones by successively attaching new vertices from K to the existing core C. The partition is considered for all $1 \leq k \leq p - 1$; for each such case, the following steps are examined:

1. **Choosing the attachment set:** There are $\binom{p}{k}$ ways to choose K.
2. **Building the core tree:** The subgraph on C must be a rooted directed tree on $p - k$ vertices, which can be done in T_{p-k} ways.
3. **Attaching the attachment vertices:** Each vertex in K must attach to a vertex in C. Since $|C| = p - k$, there are $(p - k)^k$ such assignments.

Figure 4.5 shows how the attachment of these vertices works. Without further correction, this construction results in overcounting, as the same tree may arise from multiple partitions. To ensure that only unique directed trees are counted, specifically those in which each attachment vertex has a directed path to the root via the core, the inclusion-exclusion principle is applied.



(a) The vertex set is split into a *core* (gray, forming a rooted tree at v_1) and an *attachment set* (red) to be connected to the core. Dashed arrows show possible attachment options for each $v_i \in K$.



(b) A valid directed tree where each vertex in the attachment set has a directed path to the root via the core. Inclusion–exclusion removes overcounted trees.

Figure 4.5: Recursive construction of rooted directed trees by partitioning the vertex set into a core and an attachment set, illustrating the attachment of new vertices to an existing rooted directed tree.

Necessity of the Inclusion–Exclusion Principle

Exact enumeration requires correction for this overcounting, which is achieved by applying the inclusion-exclusion principle. For each possible size k of the attachment set, the sign of the corresponding term in the summation is alternated, thus eliminating configurations that are counted multiple times due to shared attachment vertices. This yields the following fundamental recurrence relation:

Theorem 4.2.3 (Recursive Enumeration of Labeled Rooted Directed Trees)**.** *The number T_p of labeled rooted directed trees on p vertices satisfies*

$$T_p = \sum_{k=1}^{p-1} (-1)^{k-1} \binom{p}{k} (p-k)^k T_{p-k}, \qquad \text{with} \qquad T_1 = 1. \tag{4.5}$$

Here, for each $1 \leq k \leq p-1$, the term $\binom{p}{k}(p-k)^k T_{p-k}$ counts all ways of choosing an attachment set of size k, building a rooted directed tree on the remaining $p-k$ vertices, and attaching each vertex in K to the core. The alternating sign, arising from the inclusion–exclusion principle, ensures that only distinct trees are counted exactly once.

Interpretation of the Recurrence

Each term in the recurrence has a clear combinatorial interpretation:

$\binom{p}{k}$	Choose the k attachment vertices $K \subset V$
T_{p-k}	Construct the rooted directed tree on the core C
$(p-k)^k$	Connect each vertex in K to a vertex in C
$(-1)^{k-1}$	Correct for overcounting via inclusion–exclusion

Results

Figure 4.6 presents a comparison between the enumeration of rooted directed trees obtained via Robinson's recurrence and the corresponding closed-form expression. The results are identical for all $p \leq 35$, which validates the correctness of the recursive enumeration algorithm. Moreover, the figure illustrates that rooted trees, while structurally simpler than general DAGs, already show exponential growth in the number of configurations as the number of vertices increases.

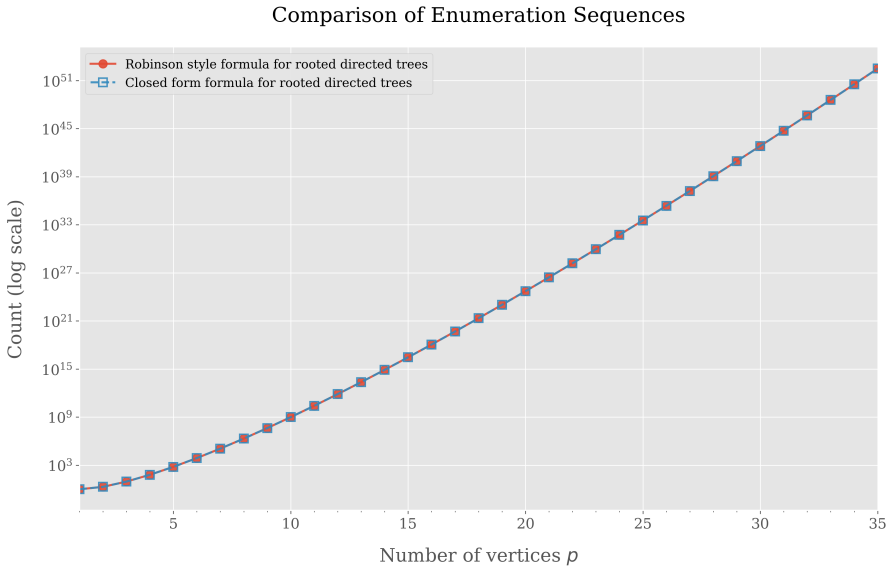


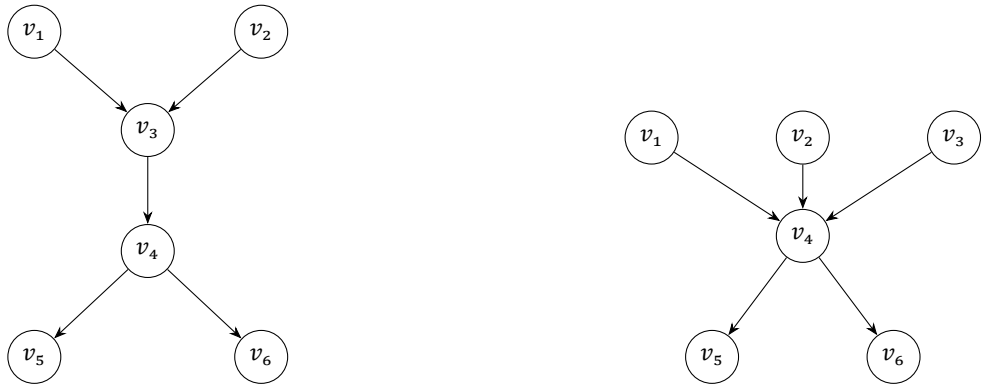
Figure 4.6: Comparison of closed form and Robinson style formula

4.3. Enumeration of Polytrees

Polytrees form an important class of directed acyclic graphs, characterized by the property that their underlying undirected structure is a tree. Unlike rooted directed trees, polytrees can contain vertcies with multiple parents, as long as there are no directed cycles present. In Figure 4.7 two example polytrees are given. This increased structural flexibility makes polytrees significant in applications such as probabilistic graphical models and Bayesian networks. In this chapter, two enumeration approaches are explored for labeled polytrees, beginning with the classical closed-form result and later deriving a recursive formula inspired by Robinson's inclusion-exclusion method.

4.3.1. Closed Form Enumeration of Polytrees

Polytrees allow nodes to have multiple parents as long as no directed cycles are present. This added flexibility means that polytrees are important in modeling hierarchical or causal structures, especially in Bayesian networks. Before examining recursive enumeration methods, the closed-form solution for counting labeled polytrees is presented, with connections to classical results in tree enumeration.



(a) Polytree where vertex v_3 has two parents.

(b) Polytree where vertex v_4 has three parents.

Figure 4.7: Two examples of labeled polytrees with multiple parent nodes.

Derivation of Closed Form Formula

Step 1 – Choose an undirected skeleton. By ignoring the edge directions, each polytree corresponds to a labeled undirected tree. Conversely, any such tree can serve as the skeleton of a polytree. By Cayley's formula:

$$|\text{number of labeled skeletons}| = \quad U_p = p^{p-2}.$$

Step 2 – Orient the edges. An undirected tree with p vertices contains exactly $p-1$ edges and no cycles. Since orienting edges in any way cannot introduce cycles, every edge has two possible orientations, and every assignment yields a valid DAG:

$$|\text{Orientations per skeleton}| = \quad 2^{p-1}.$$

Step 3 – Multiply independent choices. The skeleton and orientation steps are independent: each labeled tree gives rise to 2^{p-1} distinct orientations, and each combination yields a unique labeled polytree.

Closed Form Enumeration of Polytrees

Let P_p denote the number of labeled polytrees on the vertex set $V = \{1, 2, \ldots, p\}$. Recall that a polytree is a directed acyclic graph whose underlying undirected structure is a tree.

Theorem 4.3.1 (Enumeration of Labeled Polytrees)**.** *The number of labeled polytrees on p vertices is given by*

$$P_p = 2^{p-1}\, p^{p-2}, \qquad \text{for } p \geq 1. \tag{4.6}$$

In this formula, the factor p^{p-2} counts the number of labeled undirected trees on p vertices, as given by Cayley's theorem, while the factor 2^{p-1} accounts for the possible orientations of the $p-1$ edges in each tree. Each undirected tree admits exactly 2^{p-1} acyclic orientations, and each orientation produces a unique labeled polytree.

4.3.2. Enumeration of Polytrees Using Robinson's Argument

Although Cayley's formula yields an exact count of labeled polytrees, the objective here is to derive a recursive formula for P_p that uses the inclusion-exclusion technique introduced by Robinson for the enumeration of acyclic digraphs [10]. This recursive form not only offers structural insight, but also illustrates the power of inclusion–exclusion in graph enumeration.

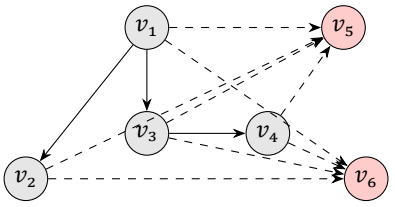
Construction via Core and Attachment Set

The derivation of the recurrence begins by partitioning the vertex set $V = \{1, \dots, p\}$ into two disjoint subsets [2]:

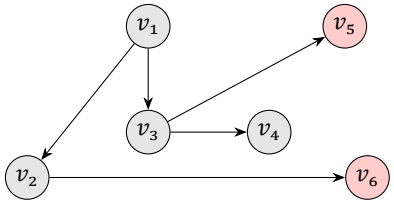
- A *core* C of size $p - k$, which forms a polytree. The core represents the portion of the structure that is already assembled; it is a connected, acyclic subgraph containing the root and serves as the foundation to which new vertices may be attached.
- An *attachment set* K of size k, consisting of vertices that must connect to the core. These are the vertices not yet incorporated into the polytree. Each vertex in K will be joined to a node in the core, thereby expanding the polytree while preserving its acyclic and connected nature.

This decomposition facilitates a recursive construction: by iteratively attaching vertices from the attachment set K to the existing core C, all polytrees of size p can be generated from smaller instances, ultimately resulting in the desired recurrence relation. All possible values $1 \leq k \leq p-1$ are considered. For each partition of this form, the construction proceeds as follows:

1. **Choosing the attachment set:** There are $\binom{p}{k}$ ways to choose the set $K \subset V$.
2. **Building the core tree:** The subgraph on $C = V \setminus K$ must form a labeled polytree on $p - k$ vertices. This can be done in P_{p-k} ways.
3. **Attaching the vertices:** To maintain symmetry among all nodes, including the root, the root vertex $r \in C$ is regarded as one of the attachment points. Consequently, there are $k + 1$ vertices (comprising the k attachment vertices in K together with the root r) that must each select a parent in the core C. Since $|C| = p - k$, there are $(p - k)^{k+1}$ ways to assign these parent connections. These connections maintain acyclicity because all edges are directed into the core. This step is shown in figure 4.8.
4. **Orienting the edges:** Each of the k attachment edges can be oriented in two ways (either toward or away from the core), resulting in 2^k possible orientation choices.
5. **Correcting for overcounting:** This construction counts each labeled polytree p times, corresponding to every possible choice of root. Therefore, a division by p is required to obtain the correct enumeration.



(a) Partition into core (gray) and attachment set (red). The root v_1 is part of the core but also participates as one of the $k + 1$ attachment endpoints. Each red vertex independently selects a parent in the core.



(b) An example of a valid configuration: each attachment vertex selects a parent in C. Since each labeled polytree appears once for each possible choice of root, division by p is required to correct for overcounting.

Figure 4.8: Visualizing the $2^k(p - k)^{k+1}$ attachment factor and the final normalization in Robinson's recurrence for labeled polytrees.

Necessity of the Inclusion–Exclusion Principle

A naive construction for enumerating labeled polytrees leads to significant overcounting. This is because multiple choices of the attachment set K and their connections to the core C can yield the same polytree. To correct for this overcounting and ensure that each distinct polytree is counted exactly once, the inclusion-exclusion principle is applied. For each possible size k of the attachment set, the sign of the corresponding term is alternated, thereby systematically removing overlapping and overcounted configurations from the total. This reasoning leads to the following recurrence relation for labeled polytrees:

Theorem 4.3.2 (Recursive Enumeration of Labeled Polytrees)**.** *The number P_p of labeled polytrees on p vertices satisfies the recurrence*

$$P_1 = 1, \quad P_p = \frac{1}{p} \sum_{k=1}^{p-1} (-1)^{k-1} \binom{p}{k} \cdot 2^k \cdot (p-k)^{k+1} \cdot P_{p-k}, \qquad \text{for } p \geq 2. \tag{4.7}$$

Here, for each k, the summand $\binom{p}{k} \cdot 2^k \cdot (p-k)^{k+1} \cdot P_{p-k}$ counts the ways to select and attach k new vertices to a core polytree, including possible orientations and attachment choices. The alternating sign, as dictated by the inclusion–exclusion principle, ensures that only valid and distinct polytrees are counted.

Interpretation of the Recurrence Terms

Each term in the recurrence (4.7) corresponds to a meaningful combinatorial step:

$\binom{p}{k}$	Choose the k attachment vertices $K \subset V$
P_{p-k}	Build a polytree on the core $C = V \setminus K$
$(p-k)^{k+1}$	Choose parent nodes in the core for k attachment vertices and the root
2^k	Assign a direction to each of the k attachment edges
$(-1)^{k-1}$	Alternate signs to apply inclusion–exclusion
$\frac{1}{p}$	Normalize to count each labeled polytree exactly once

Results

Figure 4.9 compares the enumeration of labeled polytrees using two approaches: the Robinson-style recurrence-based method and the closed form expression derived from Cayley's formula. The values are plotted on a logarithmic scale. As shown, both curves are identical up to $p = 35$, with the Robinson-style count matching the closed form expression. This confirms the accuracy of the implementation of the recurrence and supports the known result.

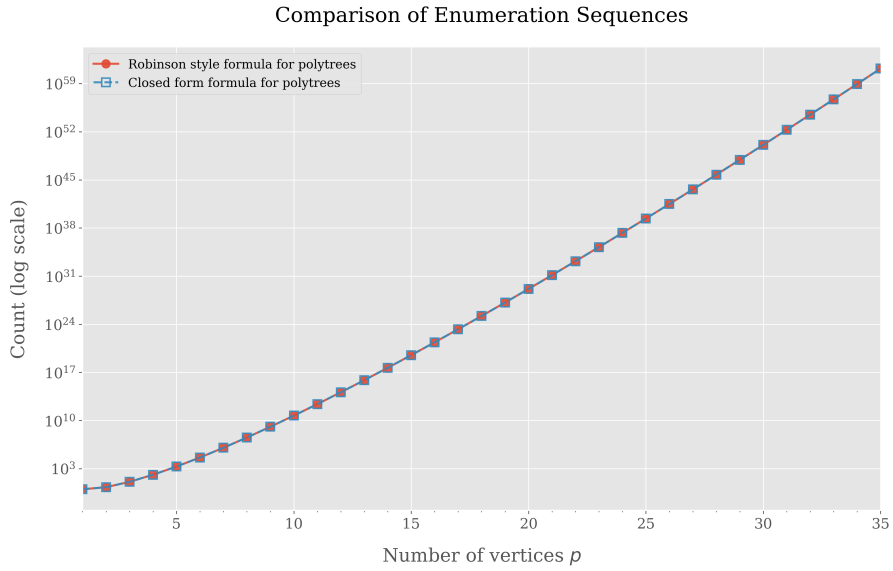


Figure 4.9: Comparison of closed form and Robinson style formula

4.4. Enumeration of Triangle-Covered DAGs

Among the many possible structural constraints on directed acyclic graphs, one particularly intriguing family is formed by triangle-covered DAGs, graphs in which every edge belongs to exactly one triangle, and each triangle is oriented to preserve acyclicity. These structures provide a rich setting for

exploring many different types of graphs and demonstrating the adaptability of combinatorial enumeration techniques. In this chapter, both closed-form and recursive approaches for counting such labeled triangle-covered DAGs are presented, revealing new insights into their combinatorial complexity.

4.4.1. Closed Form Enumeration of Triangle-Covered DAGs

Triangle-covered DAGs form a fascinating class in which every edge belongs to exactly one triangle, and all triangles are oriented to avoid cycles. Due to their rigid local structure and recursive construction, it is possible to derive an explicit closed-form expression for the number of such labeled DAGs. In this subsection, a formula for the enumeration of triangle-covered DAGs is established, together with an explanation of the underlying combinatorial principles. The objective is to derive an explicit expression for the number of such labeled DAGs on $p = 2t + 1$ vertices.

Definition

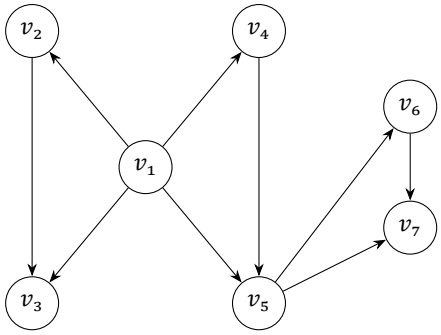
A **triangle-covered DAG** is a labeled DAG with the following properties:

- The underlying undirected graph is connected;
- Each edge is part of exactly one triangle;
- Every vertex appears in at least one triangle;
- Every triangle is oriented without forming a directed cycle;
- The entire graph is acyclic.

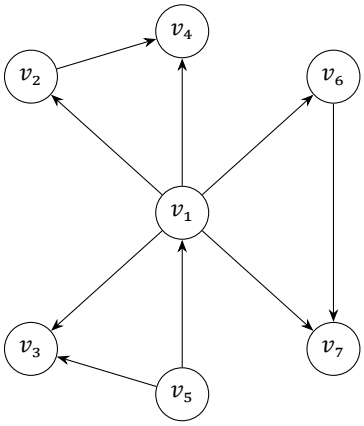
Because the triangles only intersect at shared vertices (never share edges), orienting each triangle acyclically guarantees that the entire digraph remains acyclic. Figure 4.10 shows two different triangle-covered DAG examples.

The following notation is used throughout the following section:

$t =$ number of triangles, $\quad p = 2t + 1 =$ number of vertices, $\quad m = 3t =$ number of edges.



(a) A triangular-cactus where articulation v_1 participates in two triangles and v_5 continues a chain.



(b) A compact triangular-cactus where vertex v_1 lies in three edge-disjoint triangles.

Figure 4.10: Two examples of labeled triangle-covered DAGs. In both, every vertex belongs to at least one directed triangle and the global orientation is acyclic.

Counting the Undirected Structures

Before enumerating triangle-covered directed acyclic graphs, it is necessary to first count the number of underlying undirected graphs with the desired triangular structure. Specifically, the class under consideration consists of connected labeled graphs constructed from t edge-disjoint triangles, with the property that each vertex is included in at least one triangle.

Noam D. Elkies [4] derived a closed-form enumeration formula for such labeled undirected graphs with $p = 2t + 1$ vertices:

Theorem 4.4.1 (Enumeration of Labeled Triangle-Covered Graphs)**.** *The number T_p of labeled, connected, undirected graphs on $p = 2t + 1$ vertices, in which every edge is contained in exactly one triangle and every vertex is part of at least one triangle, is given by*

$$T_p = p^{\frac{p-3}{2}} \cdot (p-2)!! = (2t+1)^{t-1} \cdot (2t-1)!!, \qquad \text{where } p = 2t+1. \tag{4.8}$$

Here, $(2t-1)!!$ denotes the double factorial, which can be written in closed form as

$$(2t-1)!! = \frac{(2t-1)!}{2^{t-1}(t-1)!}.$$

Counting Acyclic Orientations

Next, the number of ways to assign directions to the edges of each triangle while avoiding directed cycles is determined. There are $2^3 = 8$ total orientations for the three edges in a triangle, but only 6 of these are acyclic. Since triangles are edge-disjoint and only intersect at most in a single vertex, each triangle can be oriented independently. Consequently, each graph has 6^t valid acyclic orientations. Multiplying this quantity by the undirected count T_p yields the total number of labeled triangle-covered DAGs:

$$A_p = 6^t \cdot T_p = 6^{\frac{p-1}{2}} \cdot p^{\frac{p-3}{2}} \cdot (p-2)!! = 6^t \cdot (2t+1)^{t-1} \cdot (2t-1)!!$$

In summary, the structure of triangle-covered DAGs allows for the derivation of an explicit closed-form enumeration formula by combining results from undirected graph enumeration and acyclic orientation counting. The resulting expression highlights the interplay between local motifs and global acyclicity, which provides a foundation for further analysis and serves as a benchmark for verifying the Robinson method of enumeration.

4.4.2. Enumeration of Labeled Triangle-covered Graphs Using Robinson's Argument

Although the closed-form formula is exact and efficient to compute, a recursive formula is now derived using the inclusion-exclusion technique introduced by Robinson in his work on acyclic digraphs [10]. This method gives deeper structural insight into how these graphs are built and demonstrates how local constraints (such as the triangle structure) can be incorporated recursively.

Key Idea: Decomposition by Leaf-triangles

Every triangular-covered or triangular-cactus graph contains at least one leaf-triangle, a triangle that connects to the rest of the graph via a single shared vertex (called the articulation point). The two remaining vertices of the triangle are degree-2 leaf vertices, each belonging to only that triangle.

This observation leads to an inclusion–exclusion approach. Let:

- $V = \{1, 2, \ldots, p\}$ be the labeled vertex set;
- For each $v \in V$, define the property $\mathcal{P}_v$: "v is a leaf vertex."

Since every cactus must contain at least one leaf-triangle, there must exist at least two leaf vertices. Consequently, no cactus graph avoids all properties $\mathcal{P}_v$, and the inclusion-exclusion principle can be applied to the sets of vertices that are required to be leaves.

Recursive Decomposition Strategy

A subset $S \subseteq V$ of leaf vertices is fixed and the number of triangular-cactus graphs in which all vertices in S are leaf vertices is counted. This quantity is nonzero only when $|S| = 2s$, as each leaf-triangle contributes exactly two leaf vertices. The construction works as follows:

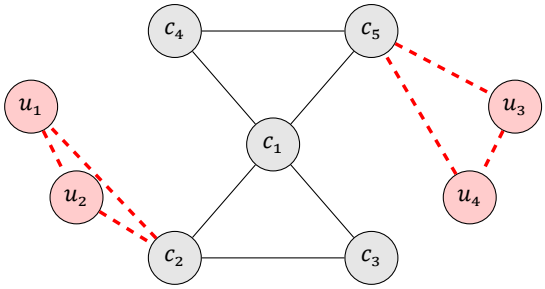
1. **Choose the leaf vertices:** There are $\binom{p}{2s}$ ways to choose a set $S \subseteq V$ of size $2s$.
2. **Pair the leaf vertices:** The vertices in S must be grouped into s unordered pairs (so that each pair becomes the two leaf vertices of a leaf-triangle). The number of different ways to do this is

$$\frac{(2s)!}{2^s s!}.$$

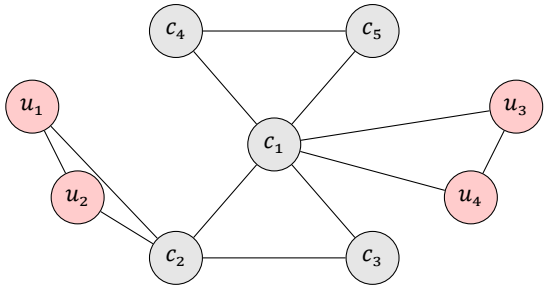
This formula comes from a simple idea: First, list all $2s$ vertices in any order. There are $(2s)!$ such arrangements. Then, group the list into consecutive pairs. However, this counts each unordered pairing more than once, because swapping the order inside any pair doesn't change the pairing (so divide by 2 for each of the s pairs,2^s in total). The s pairs themselves can be listed in any order, but as the pairs are unordered, we must divide by $s!$.

3. **Attach the leaf-triangles to the core:** For each pair of leaf vertices, the attachment is made to an articulation point in the core. As the core contains $p - 2s$ vertices and the selections are made independently with repetition allowed, there exist $(p - 2s)^s$ possible choices for assigning articulation points.

4. **Apply inclusion–exclusion:** For each s, the sign alternates as $(-1)^{s+1}$ due to the inclusion–exclusion principle.



(a) Each unordered leaf pair may attach to *any* core vertex, giving the factor $(p - 2s)^s$.



(b) A particular choice of articulation vertices attaches the two leaf-triangles, producing a cactus on $p = 2t + 1$ vertices.

Figure 4.11: Attachment of leaf-triangles: The core (gray) is itself a triangular-cactus (two edge-disjoint triangles sharing c_1). (a) Each dashed red pair indicates one of $(p-2s)^s$ possible attachments of the leaf-triangles. (b) After selecting specific articulation points (c_2 for $\{u_1, u_2\}$ and c_1 for $\{u_3, u_4\}$) we obtain the full cactus required in step 4 of the decomposition.

Final Recursive Formula

Combining the construction steps and applying the inclusion–exclusion principle yields the following recurrence relation for the number of labeled triangular-cactus graphs:

Theorem 4.4.2 (Recursive Enumeration of Labeled Triangular-Cactus Graphs)**.** *Let C_t denote the number of labeled connected undirected graphs composed of t edge-disjoint triangles such that every vertex is contained in at least one triangle, with $p = 2t + 1$ vertices. Then C_t satisfies the recurrence:*

$$C_1 = 1, \quad C_t = \sum_{s=1}^{t} (-1)^{s+1} \binom{2t+1}{2s} \frac{(2s)!}{2^s s!} (2t + 1 - 2s)^s C_{t-s}, \qquad \text{for } t \geq 2. \tag{4.9}$$

Here, each term in the sum corresponds to the process of selecting and removing s leaf-triangles, pairing the $2s$ leaf vertices, and reattaching them to a smaller core graph with $t - s$ triangles. The alternating sign, as dictated by the inclusion–exclusion principle, ensures that each configuration is counted exactly once and only when the chosen vertices are required to be leaf vertices.

From Triangular-Cactus Graphs to Triangle-Covered DAGs

Finally, the enumeration of labeled DAGs with a triangular-cactus underlying structure is considered. Each triangle has 3 edges, and each edge can be oriented in 2 ways, giving $2^3 = 8$ total edge orientations. However, exactly 2 of these form a directed 3-cycle, so there are $8 - 2 = 6$ acyclic orientations per triangle. Since triangles are edge-disjoint and only meet at articulation vertices, their orientations are independent. Thus, each triangular-cactus graph with t triangles yields 6^t distinct labeled DAGs through independent acyclic orientations of its triangles.

The total number of labeled DAGs with a triangular-cactus structure is therefore:

$$\text{Number of DAGs} = 6^t \cdot C_t.$$

Results

Figure 4.12 compares the enumeration of triangle-covered DAGs using a Robinson-style recurrence and a closed-form approach. The log-scale plot reveals a perfect match between the two methods. In particular, the two counts coincide exactly for odd values of p. This is because in the Robinson-style construction the recursion operates over the number of triangles instead of the number of vertices. This comparison confirms the validity of both enumeration formulas.

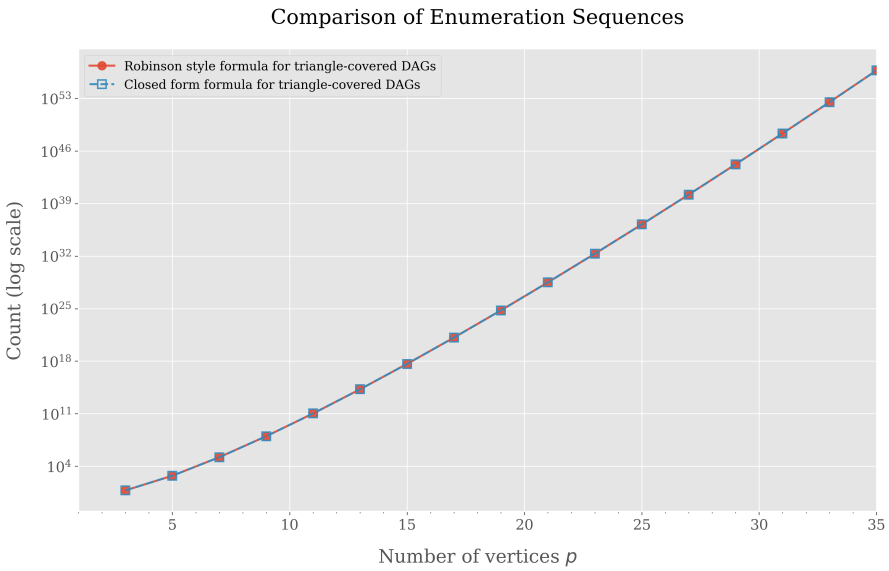


Figure 4.12: Comparison of closed form and Robinson style formula

4.5. Comparison of Enumeration of Different Types of Graphs on p Nodes

To illustrate the combinatorial explosion in the number of possible structures, Table 4.1 lists the exact counts of labeled DAGs, rooted directed trees, polytrees and triangle-covered DAGs for small values of p. As p increases, the number of possible DAGs far exceeds that of the more structurally constrained subclasses, such as trees and polytrees.

Table 4.1: Exact counts for small p ($1 \leq p \leq 7$) of various labeled acyclic structures.

p	Rooted Directed Trees	Triangle-Covered DAGs	Polytrees	DAGs
1	1		1	1
2	2		2	3
3	9	6	12	25
4	64		128	543
5	625	540	2 000	29 281
6	7 776		41 472	3 781 503
7	117 649	158 760	1 075 648	1 138 779 265

Figure 4.13 further visualizes the growth of each enumeration sequence on a logarithmic scale. The figure demonstrates that, while all classes increase rapidly, the number of general DAGs grows much more quickly than any of the structured subclasses. For example, with $p = 7$, the number of labeled DAGs exceeds one billion, while the counts for rooted trees and polytrees are several orders of magnitude smaller.

The exponential growth in these sequences emphasizes the difficulty of enumeration and highlights the significance of analytical methods and recursive formulas for both theoretical analysis and practical applications. In particular, structural constraints such as being a tree, polytree, or triangle-covered limit the solution space dramatically, but do not fully mitigate the combinatorial explosion for large p.

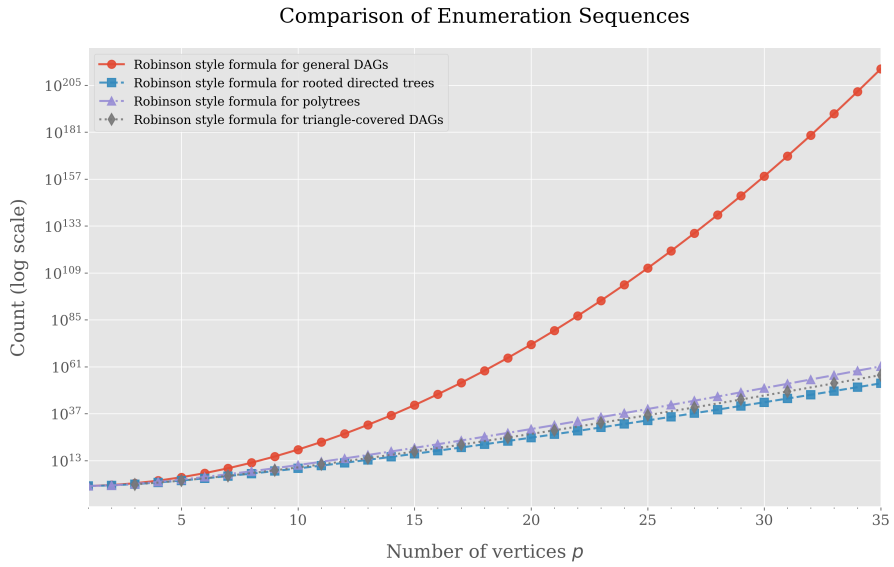


Figure 4.13: Growth comparison of labeled rooted directed trees, triangle-covered labeled DAGs, labeled polytrees, and labeled DAGs for $p \leq 35$. The vertical axis is logarithmic due to the rapid growth.

5

Discussion and Conclusion

In this report, the enumeration of labeled DAGs under a variety of structural constraints was explored. The central theme of this report is the integration and extension of recursive counting techniques, most notably the inclusion-exclusion based method introduced by Robinson, to address both classical and new enumeration problems in graph theory.

The derivation and analysis of Robinson's recursive formula serve as a foundation for understanding the exponential complexity of DAGs, while also providing a flexible framework for extensions. By partitioning graphs according to out-points and applying the inclusion–exclusion principle, we obtain a recurrence that efficiently counts labeled DAGs and can be adapted for refined constraints, such as fixing the number of arcs or out-points.

Beyond general DAGs, Robinson's approach has been successfully adapted to the enumeration of several important specialized structures, including rooted directed trees, polytrees, and triangle-covered DAGs. For each of these classes, both closed-form expressions and recursive constructions were provided. The results reveal combinatorial connections between these special classes and highlight how local attachment rules can encode a wide range of global structural properties. The closed-form results provide important checks and context for the recursive methods, reinforcing their validity.

A key observation is the rapid growth in the number of possible structures as the number of vertices increases. This occurrence, illustrated through tables and plots, underscores the combinatorial explosion faced in practical applications, such as Bayesian network structure learning, scheduling, and network design. Although the considered extensions restrict the enumeration and thus result in substantially lower counts compared to general DAGs, the enumeration for rooted directed trees, while the lowest among the examined classes, remains very large as the number of vertices grows. While closed-form and recursive formulas provide theoretical insight and allow efficient computation for small graphs, enumeration quickly becomes infeasible for larger instances.

Despite these advances, several challenges and opportunities remain for future work. Other possible directions include the enumeration of DAGs with more complex constraints (such as connectivity or bounded degree), the study of asymptotic properties, and the development of efficient algorithms and software implementations for practical use.

In conclusion, this thesis brings together a variety of enumeration problems within a single recursive framework, offering both new formulas and theoretical insights. The results contribute to a deeper understanding of the structure and complexity of directed acyclic graphs and provide a solid basis for further study in combinatorial theory and its applications across science and engineering.

6

Future Work

The results presented in this thesis lay the foundation for a wide range of future research directions in the combinatorial analysis of DAGs and related graph structures. Several promising directions are described below.

Enumeration of Larger Cycle Structures

Although this thesis has focused on acyclic graphs and local structures such as triangles, an important challenge for future research is the enumeration of graphs containing larger cycles. Understanding the enumeration of larger cycles is not only a natural extension but is also important for modeling real-world networks, where feedback loops and cyclical dependencies play a crucial role. Developing recursive or inclusion-exclusion-based methods for these classes may yield new insights but will involve complexity due to global and local constraints.

Importantly, enumeration of labeled DAGs with underlying graphs built from larger cycles (such as squares, pentagons, or any k-cycle) can be approached using a generalization of the Robinson-style inclusion–exclusion method used for triangles. The main idea is that, as with triangles, you can systematically choose leaf cycles, cycles that attach to the rest of the graph at a single point, and account for all the ways to reattach them. For each larger cycle, you consider the possible ways to assign its extra vertices, group them together, and choose an articulation vertex where the cycle attaches. The number of ways in which each cycle can be acyclically oriented can be calculated independently for each block and multiplied together in the final count.

Although the enumeration problem changes in the presence of larger cycles, the strategy remains unchanged: the inclusion-exclusion principle enables the enumeration of all possible configurations by successively adding and removing substructures while correcting for overcounting. This approach paves the way for the application of recursive and algebraic techniques to a much broader class of directed graph structures.

Combining Enumeration Techniques

Another extension is the combination and generalization of the techniques explored in this thesis. For instance, hybrid approaches may allow for efficient enumeration of graphs with multiple simultaneous constraints. Enumeration of labeled DAGs for hybrid structures, for example, polytrees where a certain number of triangles are attached as additional building blocks is also possible. In such models, the construction can begin with a tree-shaped core to which additional triangles are subsequently attached at various locations. Each triangle is connected to a tree node and introduces two new vertices, expanding the structure in a controlled, non-branching way.

The Robinson-style argument extends naturally to this setting: after fixing the underlying tree, the inclusion-exclusion principle is applied to systematically account for all possible subsets of the newly added vertices that may be designated as leaves. At each step count all the ways the original structure can be created and then count the number of possibilities to attach these triangles to the original tree, keeping track of labelings and attachment points. Finally, the possible orientations for each triangle are counted independently, multiplying the total by a constant.

This approach means that you can systematically enumerate very complex classes of labeled DAGs by breaking them into manageable steps: first, count the core structure, then count the ways to attach and orient additional subgraphs. It also makes it feasible to extend the enumeration to networks with even more intricate local or global properties, such as combinations of cycles, trees, and other types of graphs.

Enumeration of Labeled DAGs Without V-structures

A major source of combinatorial complexity in DAGs arises from V-structures, which are formed when two edges from different sources point to the same node. However, there are important classes of DAGs, such as modeling sequential processes, chains of causality, or certain types of molecular graphs, where such V-structures are forbidden by design [8].

For example, in the case of triangle-covered DAGs, we can consider sequences of triangles connected end-to-end (triangle chains). The key advantage of this structure is that it allows for a systematic recursive decomposition: at each step, one can remove an end-triangle, count the ways to orient and label the remaining smaller chain, and then reattach the triangle in only a fixed number of ways. This process can be modeled using a variant of the Robinson-style inclusion–exclusion principle, which alternates between adding and subtracting counts based on which vertices are forced to be leaves. The orientation of each triangle can be accounted for independently, allowing the total count to be constructed from the product of local choices and the recursive structure of the chain.

Overall, it is plausible that this approach may generalize to other classes of block graphs where V-structures are forbidden, potentially providing a useful technique for enumerating DAGs with sequential chain-like dependencies. Such models are particularly relevant in fields where order, sequence, or non-branching flows are fundamental, and the counting methods developed here could serve as a template for addressing more complex restrictions in future research.

Importance and Broader Impact

The capability to enumerate these advanced graph classes is significant beyond purely theoretical considerations. It directly impacts our ability to model, analyze, and optimize complex systems in fields ranging from artificial intelligence and biology to operations research and communication networks. As such, future work on the enumeration of larger cycles, and V-structures, potentially by combining and generalizing the methods presented in this thesis, will contribute both to the depth of combinatorial mathematics and to the effectiveness of applied modeling in science and engineering.

Bibliography

[1] Arthur Cayley. “A Theorem on Trees”. In: *Quarterly Journal of Pure and Applied Mathematics* 23 (1889), pp. 376–378.

[2] Yi Chen, Arthur Choi, and Adnan Darwiche. “On Pruning with the MDL Score”. In: *Proceedings of the 8th International Conference on Probabilistic Graphical Models (PGM)*. Ed. by Antoni Bayes, Kristian Kersting, and José Miguel Hernández-Lobato. Vol. 52. JMLR Workshop and Conference Proceedings, 2016, pp. 64–75.

[3] Reinhard Diestel. *Graph Theory*. 5th ed. Electronic edition freely available. Springer, 2017.

[4] Noam D. Elkies. “Enumeration of Undirected Labeled Triangle-Covered Graphs”. Manuscript, December 16, 2002. Dec. 2002.

[5] Sharon B. Gillispie and Michael D. Perlman. “Enumerating Markov equivalence classes of acyclic digraph models”. In: *Annals of Statistics* 29.2 (2001). https://arxiv.org/pdf/1301.2272, pp. 548–566.

[6] Michael S. Haigh and David A. Bessler. “Causality and Price Discovery: An Application of Directed Acyclic Graphs”. In: *The Journal of Business* 77.4 (2004), pp. 1099–1121. DOI: 10.1086/422632. URL: https://doi.org/10.1086/422632.

[7] Guy Melançon et al. “Uniform generation of random DAGs for Bayesian networks”. In: *arXiv preprint* arXiv:1202.6590 (2012). URL: https://arxiv.org/abs/1202.6590.

[8] Bryan Pardo. *DAG Discovery and Testing*. https://bpb-us-w2.wpmucdn.com/u.osu.edu/dist/7/36891/files/2017/07/DAGDiscoveryTesting-20986mr.pdf. Accessed: 2025-06-22. 2017.

[9] Daniel Ralaivaosaona, Nyandorirak Rasendrahasina, and Stephan Wagner. “The Probability that a Random Digraph is Acyclic”. In: *31st International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA 2020)*. Vol. 159. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 25:1–25:17. DOI: 10.4230/LIPIcs.AofA.2020.25. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12080/.

[10] R. W. Robinson. *Counting Unlabeled Acyclic Digraphs*. Unpublished manuscript. Retrieved from scanned copy provided in TU Delft project archive. 1977.

[11] V. I. Rodionov. “On the number of labeled acyclic digraphs”. In: *Discrete Mathematics* 105.1-3 (1992), pp. 319–321.

[12] Shivani Sachdeva and Poonam Panwar. “A Review of Multiprocessor Directed Acyclic Graph (DAG) Scheduling Algorithms”. In: *International Journal of Computer Science and Communication* 6.1 (2015), pp. 67–72. URL: https://www.researchgate.net/publication/280978306_A_Review_of_Multiprocessor_Directed_Acyclic_Graph_DAG_Scheduling_Algorithms.

[13] Hirotaka Tamanoi. *Graphical Enumeration*. https://tamanoi.math.ucsc.edu/tamanoi/GraphicalEnumeration.pdf. Accessed: April 29, 2025. 2004.

[14] Douglas B. West. *Introduction to Graph Theory*. 2nd. Upper Saddle River, NJ: Prentice Hall, 2001. ISBN: 9780130144003.