

ON THE ENUMERATION OF ALL UNIQUE PATHS OF RECOMBINING TRINOMIAL TREES*

ETHAN TORRES[†], RAMAVARAPU SREENIVAS[‡], AND RICHARD SOWERS[§]

Abstract. Recombining trinomial trees are a workhorse for modeling discrete-event systems in option pricing, logistics, and feedback control. Because each node stores a state-dependent quantity, a depth- D tree naïvely yields $\mathcal{O}(3^D)$ trajectories, making exhaustive enumeration infeasible. Under time-homogeneous dynamics, however, the graph exhibits two exploitable symmetries: (i) translational invariance of nodes and (ii) a canonical bijection between admissible paths and ordered tuples encoding weak compositions. Leveraging these, we introduce a **mass-shifting enumeration algorithm** that slides integer “masses” through a cardinality tuple to generate exactly one representative per path-equivalence class while implicitly counting the associated weak compositions. This trims the search space by an exponential factor, enabling markedly deeper trees—and therefore tighter numerical approximations of the underlying evolution—to be processed in practice. We further derive an upper bound on the combinatorial counting expression that induces a theoretical lower bound on the algorithmic cost of $\sim \mathcal{O}(D^{1/2} 1.612^D)$. This correspondence permits direct benchmarking while empirical tests, whose pseudo-code we provide, corroborate the bound, showing only a small constant overhead and substantial speedups over classical breadth-first traversal. Finally, we highlight structural links between our algorithmic/combinatorial framework and Motzkin paths with Narayana-type refinements, suggesting refined enumerative formulas and new potential analytic tools for path-dependent functionals.

Key words. Discrete Mathematics, Combinatorial Trees, Graph Algorithms, Algorithmic Complexity, Enumeration, Option Pricing

AMS subject classifications. 05C05, 05C85, 05A15, 68Q25, 68R10, 68Q17

1. Introduction. Recombining trinomial trees themselves exist as an approximation of a branching process. This is apparent since it is not always true that a process will return to a previous value after evolving to one in the future. However, these structures are well researched as approximations in the context of discrete event control problems or option pricing where approximations such as these are more strongly held [4, 6, 7]. With that aside, all trees, whether of a recombining nature or not, have huge exponential blow-ups in time and spatial complexity as they evolve. In particular, as the recombining trinomial tree evolves, it quickly generates trillions of distinct paths, whose explosion can only be controlled by algorithms and clever use of data structures. This problem has been thoroughly studied, most famously in Knuth’s series on combinatorial algorithms [14, 15].

In this paper, we aim to exploit the graphical structure of the tree, in order to avoid the combinatorial explosion typically associated with full path enumeration. This is made possible by using structural symmetries. More specifically, we use the property of translational equivalence among nodes, where identical values persist in depth shifts, preserving accumulated quantities along different paths. Consider the example shown in Figure 1. The recombining tree consists of 25 nodes, each identified by a pair of integers: *depth* (ranging from 0 at the root to 4 at the terminal level) and *position* (indicating horizontal placement). Values are assigned purely by position,

*Submitted to the editors Oct. 3.

[†]Department of Industrial Engineering, University of Illinois, Urbana-Champaign, IL (ethanjt2@illinois.edu).

[‡]Department of Industrial Engineering, University of Illinois, Urbana-Champaign, IL (rsree@illinois.edu).

[§]Department of Industrial Engineering; Department of Mathematics, University of Illinois, Urbana-Champaign, IL (r-sowers@illinois.edu).

so all nodes at the same position share the same value. For instance, the root at position 0 has value 20, while its children at positions -1, 0, and 1 take values 18, 20, and 22. Paths then accumulate these position-based values from root to terminal nodes. In effect this is just a histogram representation—the node values themselves are illustrative and not essential.

Let’s take a look at [Figure 1](#) and consider the terminal node ending at position 0. The blue, green, and red paths in [Figure 1](#) each sum to 102. Each path visits the nodes with value 22 once, and visits the node with values 20 4 times; the sum is

$$(1.1) \quad 4 \times 20 + 1 \times 22 = 102.$$

Generally, our setup allows for multiple paths to have the same accumulated value.

Trees are often used to discretize path integrals. In our case, we want to average the path-sum of values (e.g., terms like 102) over all paths terminating at position 0. A naïve method would enumerate every such path, compute its path-sum, and then average—quickly becoming infeasible as the tree depth grows. Instead, one can enumerate all *possible* path-sum values and count how many paths realize each. For example, in [Figure 1](#) there are three colored terms corresponding to position 1 once and position 0, so $N_{102} = 3$. The average is then obtained by summing $p N_p$ (e.g. 3×102) over all path-sum values p and normalizing by $\sum N_p$:

$$\text{Average} = \frac{\sum_p p N_p}{\sum_p N_p}.$$

This amounts to a Lebesgue-style integral—summing “values $\times$ measures”—rather than a Riemann-style sum over base points. This shift in viewpoint dramatically reduces computation by replacing an enumeration over exponentially many paths with an aggregation over the typically far smaller set of distinct path-sum values.

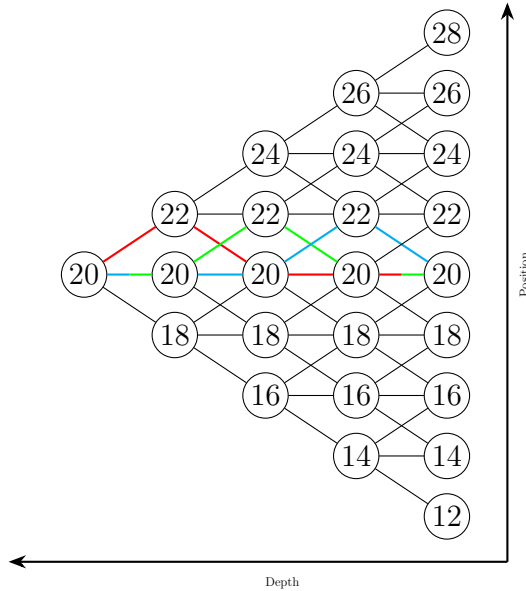


Fig. 1: Recombining Tree with Values

Note that in this figure we explicitly display the value stored at each position. In every other figure, we omit these values as they are implicitly present, and—under the labeling scheme introduced shortly—each node retains the same value throughout. Suppressing them in later diagrams keeps the visual presentation clear and uncluttered. We now introduce our notation for a tree of maximum depth D , $\mathcal{T}_D$, and informally define the set of all paths it generates as $\mathcal{P}$. We also define an informal reconstruction function $\mathcal{C}(\mathcal{P})$, where $\mathcal{C}(\mathcal{P})$ denotes all the possible combinations of paths, which constructs the tree by taking the union of all paths—i.e., $\mathcal{T}_D = \mathcal{C}(\mathcal{P})$. This is formally addressed later in [Section 2](#).

This relation highlights that the full tree is simply the collection of all possible path combinations. Due to the recombining nature of the tree, many of these paths are structurally redundant. That is, multiple paths contain the same vertices in different orders without altering the final path-dependent quantity. We refer to such redundancies as permutations of paths, denoted $\mathcal{P}(\mathcal{P})$. These permutations arise when vertex positions differ across depths but represent the same cumulative state. Hence, the set of unique paths in the tree can be informally defined as:

$$(1.2) \quad \mathcal{U}(\mathcal{P}) = \mathcal{C}(\mathcal{P}) - \mathcal{P}(\mathcal{P}).$$

Although trivially computing all paths and then subtracting duplicates could be done, such an approach fails to leverage the underlying invariance in [Figure 1](#) and still incurs the full computational cost of path enumeration. In contrast, our algorithm offers a novel method for directly computing $\mathcal{U}(\mathcal{P})$, ensuring that we never generate permutations of previously generated paths. We achieve this by *pre-pruning* permutations during the enumeration process, dramatically reducing the computational overhead.

Although the argument here is informal, it illustrates the core contribution of our method: a principled way to generate only the distinct path structures in a recombining tree, without redundancy. This offers substantial advantages in discrete event control problems and other applications involving path-dependent dynamics where recombining trees are used to model system evolution. Our approach draws on decades of work aimed at accelerating path enumeration, including the algorithms of Eades and McKay [8], Ehrlich’s loop-less generation technique [9], and the Steinhaus–Johnson–Trotter algorithm originally presented by Johnson [13]. We also refer to work by Ruskey and Williams [21] and Cheng [3] for more recent work in this particular field.

The question then arises, why specifically use a trinomial tree? A recombining trinomial lattice improves on the classic binomial grid by adding a “stay-put” or “no-change” branch, delivering countless benefits. With two free probabilities per step, it can match both the drift and the variance of the underlying process, giving second-order weak accuracy instead of the binomial’s first-order, so far fewer time steps - and hence far fewer total nodes - are needed to hit a given error tolerance [17]. The trinomial structure is also the discrete analog of a central difference scheme, which is unconditionally stable for many payoffs, in deference to the application in option pricing. This stability lets users of any algorithm that utilizes this underlying structure to stretch time increments two-to-four-fold without inducing oscillations [1]. In control problems, the three branches also line up perfectly with the canonical actions “increase, hold, decrease” eliminating the dummy states a binomial grid must invent. In summary, a third branch provides just enough freedom to hit both drift and volatility, yielding higher accuracy, greater numerical stability, cleaner boundary handling, and better scalability - all while keeping the lattice recombining.

2. Setup. We can now formally develop our definitions of a recombining rooted trinomial tree that we will use for the remainder of our analysis. To simplify later calculations, we represent this tree as a directed graph embedded in the lattice $\mathbb{Z} \times \mathbb{Z}_+$. Given a nonnegative integer D , which represents the maximum depth of the tree, we define the set of **vertices** as:

$$(2.1) \quad V_D = \{(k, d) \in \mathbb{Z} \times \mathbb{Z}_+ : 0 \leq d \leq D, |k| \leq d\}$$

and the set of directed edges as:

$$(2.2) \quad E_D = \{((k, d-1), (k+s, d)) \in V_D \times V_D : s \in \{-1, 0, 1\}\}$$

For a generic node (k, d) in V_D , we will think of d as the *depth* coordinate and k as the *position* coordinate. In addition, E_D encodes the rule that from any node at position k and depth $d-1$, we may move to depth d by stepping to $k-1, k, k+1$. The resulting graph is denoted $\mathcal{T}_D = (V_D, E_D)$, a formalization of the tree described in [Section 1](#), and is symmetric under reflection across the position axis $k = 0$, a property that will be important in later sections. We show a visualization of this graph in [Figure 2](#) to build the graphical intuition for the reader:

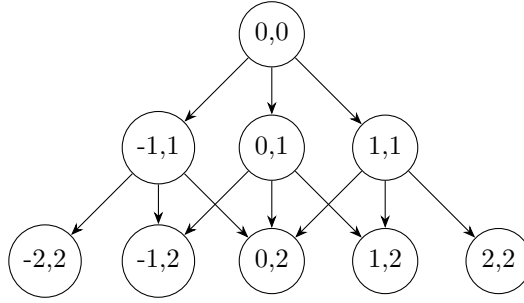


Fig. 2: $\mathcal{T}_D$ with Node-Labelings for V_D

We are interested in accumulating values along the *paths* in the tree. A path in $\mathcal{T}_D$ is a sequence $(v_0, v_1, \dots, v_D)$ of vertices such that $v_0 = (0, 0)$, each $v_d \in V_D$, and $(v_{d-1}, v_d) \in E_D$ for $d \in \{1, 2, \dots, D\}$. We provide an example of such a path in [Figure 3](#):

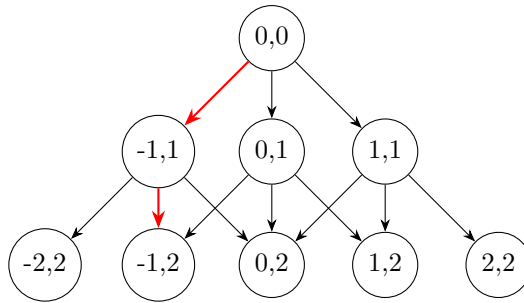


Fig. 3: Recombining Tree with Sample Path

Let $\mathcal{P}$ be the collection of all paths in $\mathcal{T}_D$. We can regard paths as walks by recording only the step increments. For each $(s_1, s_2, \dots, s_D) \in \{-1, 0, 1\}^D$, set

$$(2.3) \quad \mathbf{v}_d \stackrel{\text{def}}{=} \left(\sum_{1 \leq d' \leq d} s_{d'}, d \right), \quad d \in \{0, 1, \dots, D\},$$

with the convention that $\sum_{\emptyset} \stackrel{\text{def}}{=} 0$. Then

$$(2.4) \quad (\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_D) \in \mathcal{P}.$$

Conversely, if

$$\mathbf{p} = ((0, 0), (k_1, 1), \dots, (k_D, D)) \in \mathcal{P},$$

define the increments

$$s_d \stackrel{\text{def}}{=} k_d - k_{d-1}, \quad d \in \{1, 2, \dots, D\}.$$

This recovers the walk $(s_1, \dots, s_D) \in \{-1, 0, 1\}^D$.

Hence, the mapping between walks and paths is a bijection, and therefore

$$|\mathcal{P}| = |\{-1, 0, 1\}^D| = 3^D.$$

We can also extract the depth-indexed position sequence via $\pi : \mathcal{P} \rightarrow \mathbb{Z}^{D+1}$ defined by

$$(2.5) \quad \pi((0, 0), (k_1, 1), \dots, (k_D, D)) \stackrel{\text{def}}{=} (0, k_1, \dots, k_D).$$

3. Aggregation. Let's return to [Figure 1](#). The blue, green, and red paths all sum to 102 and end at the 20 node. We can rewrite the sums as

$$20 \times 4 + 22 \times 1 = 80 + 22 = 102,$$

reflecting the fact that 20 occurs 4 times along each path, and 22 occurs once along each path (analogous to Lebesgue integration vs Riemannian integration). Essentially, we will *index* paths on the tree to first identify the blue path, compute the sum, and then skip over the green and red paths. Doing in a way which avoids recursion, we will construct an efficient way to identify all aggregated values.

To ground the discussion, we open this section with the most straightforward technique for indexing (i.e., enumerating) every realization of a recombining trinomial tree: a depth-first recursive traversal that visits all branches until the target depth D . This classical approach was extremely important throughout our experiments because it served as a very stable baseline to check almost all of our algorithms for correctness. And while recursion does scale exponentially, this baseline serves three goals

1. its transparency made manual verification easy;
2. it provided a clean performance yard-stick for the more sophisticated algorithms introduced later; and
3. it exposed the combinatorial structure of the tree in the clearest possible way.

Our version augments the vanilla DFS [Algorithm A.1](#) with a hash-map memoisation scheme [Algorithm A.2](#) that caches the value associated with each previously visited $\{-1, 0, +1\}$ -triple; revisiting an identical sub-problem is therefore reduced to $\mathcal{O}(1)$

amortized time [18]. Although there is a slight difference in the exact cost saved due to computational time that is added from using recursion on a function and adding function calls on the stack frame, we defer that discussion to [Section 4](#).

We can then transition to analyzing the computational complexity of iterating over a trinomial tree, which we can extrapolate from [22] and our analysis in [Section 2](#).

Without memoisation the time cost is

$$(3.1) \quad \text{Time}_{\text{naïve}} = 3^D \times D = \mathcal{O}(D \cdot 3^D),$$

where the linear factor counts the computational cost associated with iterating along each D -step path. Caching removes the redundant work, leaving

$$(3.2) \quad \text{Time}_{\text{memoised}} \approx \mathcal{O}(3^D)$$

where we state that the computational time is approximate because it is constant amortized time. This classical baseline establishes the reference point for all of our later algorithms and their optimizations.

4. Removing Recursion from the Enumeration Process. Recursion — though a classical way to generate paths — is often computationally expensive. Even with careful data structures, each additional path incurs another stack frame, and total running time can grow rapidly [12]. To address this, we design a *recursion-free* algorithm that (i) exhaustively enumerates all valid paths, (ii) produces no spurious output, (iii) limits the number of loop passes (avoiding heavy overhead), and (iv) extends naturally to the unique-representative algorithm developed later in [Section 5](#).

Once paths are projected via [Equation \(2.5\)](#), it is natural to impose an order that supports a nonrecursive, “ping-pong” traversal: rather than recursing, we reset a loop cursor to a designated index and deterministically march through the next valid configuration. To set the stage, we introduce a canonical indexing by terminal node. Fix $D \in \mathbb{N}$ and, for any $k^* \in \{-D, -D+1, \dots, D\}$, define the set of all admissible depth- D paths that terminate at (k^*, D) :

$$\mathcal{P}_{k^*} \stackrel{\text{def}}{=} \{(v_0, v_1, \dots, v_D) \in \mathcal{P} : v_D = (k^*, D)\}.$$

This viewpoint lets us either sweep over all attainable k^* for a fixed depth D , or restrict to a single target k^* when only that terminal node is of interest—providing flexibility in what the generator emits.

We impose a total order via the position projection π of [Equation \(2.5\)](#). The definition is recorded as:

$$(4.1) \quad \begin{aligned} &\text{For } \mathbf{k} = (k_0, \dots, k_D), \mathbf{k}' = (k'_0, \dots, k'_D) \in \mathbb{Z}^{D+1}, \\ &\mathbf{k} \prec \mathbf{k}' \iff \exists d^* \in \{0, \dots, D\} \text{ minimal such that } k_{d^*} \neq k'_{d^*} \text{ and } k_{d^*} < k'_{d^*}, \\ &\mathbf{p} \prec_{\text{lex}} \mathbf{p}' \iff \pi(\mathbf{p}) \prec \pi(\mathbf{p}') \text{ for } \mathbf{p}, \mathbf{p}' \in \mathcal{P}. \end{aligned}$$

In practice, this ordering mirrors “walking down” the tree while updating only a small number of coordinates at each step.

For implementation, it is convenient to begin with the strictly nonnegative representatives and then add a thin post-processing layer, after each paths is generated, to recover the paths with negative excursions. Define the strictly nonnegative (“positive”) paths that end at k^* by

$$\mathcal{P}_{k^*}^+ = \{\mathbf{p} \in \mathcal{P}_{k^*} : \pi(\mathbf{p}) \in \mathbb{Z}_+^{D+1}\}.$$

We enumerate them in lexicographic order as

$$\mathcal{P}_{k^*}^+ = \{\mathbf{p}^{(n)} : n = 1, 2, \dots, |\mathcal{P}_{k^*}^+|\}, \quad \mathbf{p}^{(n)} \prec_{\text{lex}} \mathbf{p}^{(n+1)}.$$

The lexicographically *lowest* positive path is

$$\pi(\mathbf{p}^{(1)}) = (\underbrace{0, \dots, 0}_{D+1-k^*}, 1, 2, \dots, k^*).$$

The lexicographically *highest* positive path—i.e., the $\prec_{\text{lex}}$ -maximizer in $\mathcal{P}_{k^*}^+$ —has π -image

$$(4.2) \quad \begin{cases} (0, 1, 2, \dots, \frac{D+k^*}{2}, \frac{D+k^*}{2} - 1, \dots, k^*), & \text{if } D - k^* \text{ is even,} \\ (0, 1, 2, \dots, \frac{D+k^*-1}{2}, \frac{D+k^*-1}{2}, \frac{D+k^*-1}{2} - 1, \dots, k^*), & \text{if } D - k^* \text{ is odd,} \end{cases}$$

i.e., an initial monotone rise that peaks at $\lfloor \frac{D+k^*}{2} \rfloor$ and then (possibly after a single stay at the peak in the odd case) descends to k^* . This tuple serves as the *maximal seed* for our nonrecursive enumeration.

Seed-and-march (recursion-free control). Beginning from this maximal seed, [Algorithm A.3](#) advances by a simple two-step local update that preserves the admissibility rules of [Section 2](#). *Tick-down:* select the largest index j whose coordinate can be decreased by 1 without violating feasibility (i.e., `COMPUTETICKDOWN` returns j with `CHECKVALIDITY(CURRENT, j, -1) = True`); if none exists, enumeration halts. *Sweep-across:* treat the decremented unit as freed mass and greedily increment successive indices $i > j$ whenever `CHECKVALIDITY(CURRENT, i, +1)` holds, recording each valid intermediate path. This “tick-down then sweep-across” march visits the tuples of $\mathcal{P}_{k^*}^+$ in $\prec_{\text{lex}}$ order, as defined in [Equation \(4.1\)](#) using only local coordinate edits and $O(1)$ work per updated coordinate—no recursion, no whole-path comparisons. After each emitted positive path, a thin post-processing layer restores negative excursions to recover the full $\mathcal{P}_{k^*}$ without changing the control flow. This highlights the necessity of the lexicographical ordering we defined in [Equation \(4.1\)](#) to remove the recursion from the path enumeration process. In [Section 5](#) we extend this ordering to the main construction we present in this paper, making explicit how it enables the removal of recursion in these enumeration processes.

For comparison and completeness, [Algorithm A.2](#) presents a classical recursive DFS that explores children via $\{-1, 0, +1\}$. Both schemes use the same lexicographic scaffold, but the DFS incurs branch exploration and $O(D)$ call-stack depth, whereas [Algorithm A.3](#) realizes the same enumeration with deterministic tick-down (sweep-across) updates in place—effectively making “GenComb” recursion-free while preserving outputs (and, under the same ordering policy, the enumeration order). As a practical cross-check, one can hash canonical path encodings and verify that, for each (D, k^*) , the multiset of outputs from the recursive DFS matches those from our stack-free generator; implementation details appear with the algorithms.

From positive representatives to all paths. Most admissible paths reaching k^* will visit negative nodes. [Section E](#) supplies a light-weight post-processing: the *flip family* $\mathcal{F}(\cdot)$ ([Equation \(E.3\)](#)) negates any subset of unlocked positive excursions, producing valid paths with the same endpoint (which can be inserted into [Algorithm A.3](#) without adding in recursion). [Lemma E.1](#) guarantees validity and endpoint preservation, and [Proposition E.3](#) gives the exact representation [Equation \(E.5\)](#) for $k^* \geq 0$ and its sign-flipped counterpart [Equation \(E.6\)](#) for $k^* < 0$. Operationally, one may emit each

nonnegative representative as soon as it is generated by [Algorithm A.3](#), and then emit its flip family in any fixed subset order (e.g., lexicographic in the excursion indices), preserving a global total order.

Parity for seeding. If a path has j_+ up-steps, j_- down-steps, and j_0 stays, then the relations in [Equation \(F.1\)](#) and the parity constraint [Equation \(F.2\)](#) ([Section F](#)) ensure that the chosen seed obeys the even–odd compatibility between D , k^* , and j_0 . In particular, the maximal seed [Equation \(4.2\)](#) is consistent with admissible counts and enables a deterministic, recursion-free enumeration pipeline: lexicographically generate $\mathcal{P}_{k^*}^+$ ([Algorithm A.3](#)) from the maximal seed, then expand each representative by excursion flips to obtain $\mathcal{P}_{k^*}$, preserving determinism and avoiding stack-based recursion, while remaining compatible with the unique-representative machinery in [Section 5](#).

Removing Recursion from Unique Path Generation. This section forms a cornerstone of the paper’s main results. While our excursion-based treatment and the permutation of negatives across excursion blocks ([Appendix E](#)) are conceptually interesting, we relegate their detailed mechanics to the appendices to avoid obscuring the core contributions. What matters here are the ideas of *maximal paths* and *lexicographical ordering* (see [Equation \(4.1\)](#)), which not only eliminate recursion from the forthcoming algorithms but also guide the combinatorial structure underlying them. By linking “walking down” the graph to “marching through” path tuples, these tools provide intuitive evidence for the correctness of our approach. Their practical integration—via the maximal seed defined in [Equation \(4.2\)](#), the recursion-free generation [Algorithm A.3](#), and the post-processing flip mechanism justified in [Proposition E.3](#)—yields a complete, nonrecursive enumeration pipeline. With this groundwork laid and the parity constraints recorded in [Section F](#), we now proceed to the main results, confident that the reader has both a graphical and an algorithmic picture of how ordering improves and informs path enumeration.

5. Enumeration of Unique Path Combinations. As before, every $\mathbf{p} \in \mathcal{P}$ in a recombining tree $\mathcal{T}_D$ can be mapped into a tuple via [Equation \(2.5\)](#). Our goal here is to enumerate only the *unique* paths—those that differ in value, not merely by a translation or re-ordering of identical vertex positions. Although such permutations are rare near the root, they proliferate rapidly with depth, creating a large amount of redundancy. Ultimately, removing them does not alter the computational blow-up of the problem, but it *does* extend the depth and breadth of the tree we can explore, allowing finer discretisation for certain discrete event problems that require it.

Building on the classic recursive framework reviewed in [Section 2](#) and the tuple-based, recursion-free iterator enabled by our ordering scheme presented in [Section 4](#), we now construct a closed-form combinatorial count and an efficient, recursiveless algorithm that enumerates every *unique* path. The key observation is that many of the paths terminating at the same node (k^*, D) share the *same multiset* of position values and thus the same aggregated value; differing only by shifts in visit order (see [Figure 1](#)). In tuple form, this multiset is represented by a *count vector* that records how many times each position k appears. Thus we are left with the following high level algorithmic approach, which we will use this section to expand upon in detail, presenting the main results of the paper:

1. **Count–vector representation:** We treat each “cardinality tuple” as a count vector - i.e. a compact record of the multiplicities of each position in $\pi(\mathbf{p})$.
2. **Recursion-free generation:** We then iterate directly over these count vec-

tors (using the "ping-pong" iterator logic presented in [Algorithm A.3](#) in [Section 4](#)), producing *exactly one* tuple for every distinct vector and thereby eliminating shift-equivalent duplicates.

3. Natural Negative-Path Augmentation: Finally, we introduce a simple and natural augmentation which allows us to enumerate negative paths

Because of the construction of the algorithm in this manner, as will become clear in this section, duplicates are suppressed *a priori*. As a result, the tree is effectively pre-pruned while the entire procedure remains non-recursive, yielding a substantial reduction in runtime and memory consumption without an accuracy trade-off. This combined framework avoids both recursion and shift-equivalent paths, significantly enlarging the tractable region of the recombining tree.

5.1. Closed-Form Combinatorics for Unique Path Enumeration. Fix $\mathbf{p} \in \mathcal{P}_{k^*}$. The viable positions are

$$(k_-, k_- + 1, \dots, k_+ - 1, k_+)$$

where

$$(5.1) \quad \begin{aligned} k_- &= \begin{cases} \frac{k^* - D}{2} & \text{if } D - k^* \text{ is even} \\ \frac{k^* - D + 1}{2} & \text{if } D - k^* \text{ is odd} \end{cases} \\ k_+ &= \begin{cases} \frac{D + k^*}{2} & \text{if } D - k^* \text{ is even} \\ \frac{D + k^* - 1}{2} & \text{if } D - k^* \text{ is odd} \end{cases} \end{aligned}$$

For each integer k in $[k_-, k_+]$, define

$$c_k(\mathbf{p}) \stackrel{\text{def}}{=} |\{d \in \{0, 1, \dots, D\} : (\pi(\mathbf{p}))_d = k\}|;$$

$c_k(\mathbf{p})$ is the number of times that the position equals k . Combining these, we define the *cardinality tuple*¹

$$(5.2) \quad \hat{C}(\mathbf{p}) \stackrel{\text{def}}{=} (c_{k_-}(\mathbf{p}), c_{k_-+1}(\mathbf{p}), \dots, c_{k_+}(\mathbf{p})) =: \hat{c}$$

$\hat{C}(\mathbf{p})$ is the empirical count of the values taken by the path and, more specifically, $\hat{C}_{D,k^*}(\mathbf{p})$ is the empirical count of the values taken by a path that end at a particular depth D and position k^* . We formally define this mapping here:

$$(5.3) \quad \hat{C}_{D,k^*} : \mathcal{P}_{k^*} \rightarrow \mathbb{N}^{[k_-, k_+]}, \quad \hat{C}_{D,k^*}(\mathbf{p}) = (c_k(\mathbf{p}))_{k=k_-}^{k_+}.$$

We then define the full set of all *unique* cardinality tuples—that represent paths in a tree of depth D that terminate at node k^* —as $\mathcal{C}_{D,k^*}$:

$$(5.4) \quad \mathcal{C}_{D,k^*} \stackrel{\text{def}}{=} \{ \hat{C}_{D,k^*}(\mathbf{p}) = (c_k(\mathbf{p}))_{k=k_-}^{k_+} \in \mathbb{N}^{[k_-, k_+]} \}.$$

We then have that, for any $\hat{c} \in \mathcal{C}_{D,k^*}$

$$(5.5) \quad \sum_{k \in [k_-, k_+]} c_k(\mathbf{p}) = D + 1$$

¹Throughout, we write $\hat{c}$ for the image of an arbitrary path under the mapping $\hat{C}(\mathbf{p})$. This convention streamlines the exposition and avoids repeatedly carrying the full function notation in formulas and text.

This construction naturally accommodates additional weights at each node, as illustrated in [Figure 1](#). Let $\mathbf{v} : [k_-, k_+] \rightarrow \mathbb{R}$ denote any function assigning a weight $\mathbf{v}_k$ to each admissible position k (or node (k, d)). Then, for a path $\pi(\mathbf{p}) = (k_0, k_1, \dots, k_D)$, the cumulative value along the path is

$$(5.6) \quad \sum_{d=0}^D \mathbf{v}_{k_d} = \sum_{k \in [k_-, k_+]} c_k(\mathbf{p}) \mathbf{v}_k.$$

Thus the cardinality tuple provides an efficient way to aggregate weighted quantities along paths.

Before embarking on the formal construction of our algorithm and its associated combinatorial framework, we pause to introduce a sequence of key remarks. These remarks serve as the conceptual foundation of the argument: they articulate the principles that guarantee correctness and provide the scaffolding on which the later technical details rest. By presenting them explicitly at the outset, we make clear how each subsequent step of the construction is informed by—and consistent with—these foundational insights. For clarity, we present the remarks in modular form. This allows the reader to revisit them easily as the discussion unfolds, since many will be refined, extended, or specialized in later sections. In this way, the remarks serve both as a reference point and as a running thread that connects the evolving stages of the argument.

Remark 5.1 (Set splitting). Due to the nature of the argument we will begin to construct, it becomes useful to introduce notation that separates the set [Equation \(5.4\)](#) into "positive" $\hat{\mathcal{C}}$ and "mixed" $\hat{\mathcal{C}}$ (where mixed here means c_k , $k < 0$ are allowed):

$$\mathcal{C}_{D,k^*}^+ \stackrel{\text{def}}{=} \{ \hat{\mathcal{C}} \in \mathcal{C}_{D,k^*} : c_k = 0 \text{ for all } k < 0 \}, \quad \mathcal{C}_{D,k^*}^- \stackrel{\text{def}}{=} \mathcal{C}_{D,k^*} \setminus \mathcal{C}_{D,k^*}^+.$$

Thus $\mathcal{C}_{D,k^*}^+$ consists of tuples arising from paths in $\mathcal{P}_{k^*}$ that never visit $k < 0$, and $\mathcal{C}_{D,k^*}^-$ those that visit at least one negative position. Enumerating the positive part first and then augmenting the algorithm to allow negative visits yields a clean construction, without carrying along negative-index components that are identically zero in what will soon be understood as "the first phase".

Remark 5.2 (Cardinality tuples as equivalence-class keys). A central guarantee of our algorithm's correctness comes from interpreting cardinality tuples as minimal representatives of the equivalence classes of all unique paths in the tree. Each tuple indexes exactly one equivalence class, ensuring that paths are counted and ordered without duplication or omission. In this way, cardinality tuples capture precisely the minimal information required to reconstruct the tree. We therefore formalize the equivalence relation and its induced class structure here, while postponing the detailed proof to [Section G](#).

We define an equivalence relation on $\mathcal{P}_{k^*}$, using the definitions in [\(5.3\)](#) and [\(5.4\)](#) by

$$(5.7) \quad \mathbf{p} \sim \mathbf{p}' \iff \hat{\mathcal{C}}_{D,k^*}(\mathbf{p}) = \hat{\mathcal{C}}_{D,k^*}(\mathbf{p}').$$

The equivalence class of $\mathbf{p}$ is thus:

$$(5.8) \quad [\mathbf{p}]_{\sim} = \hat{\mathcal{C}}_{D,k^*}^{-1}(\hat{\mathcal{C}}_{D,k^*}(\mathbf{p})) = \{ \mathbf{p}' \in \mathcal{P}_{k^*} : \hat{\mathcal{C}}_{D,k^*}(\mathbf{p}') = \hat{\mathcal{C}}_{D,k^*}(\mathbf{p}) \}.$$

Remark 5.3 (Ordering). As discussed in Equation (4.1), lexicographic ordering is useful for implementation and best-case bounds. For cardinality tuples we *impose* lexicographic order directly on $\mathcal{C}_{D,k^*}^+ \subset \mathbb{N}^{[0,k^+]}$; via $\hat{C}_{D,k^*}$ this induces a representative-independent total order on the equivalence classes i.e. $\hat{c} \prec_{\text{lex}} \hat{c}'$. This is why it was so important to introduce the lexicographical ordering, without it, we would not be able to construct this process on equivalence classes properly. (Note that lex order on raw paths $\pi(\mathbf{p})$ need not descend to the quotient unless it is constant on the fibers of $\hat{C}_{D,k^*}$.) We will later extend the lexicographical ordering described here to all $\hat{c} \in \mathcal{C}_{D,k^*} \subset \mathbb{N}^{[k_-,k_+]}$ in Equation (5.43).

With all these remarks fully established, we can now build an enumeration of unique elements of $\mathcal{P}_{k^*}$ organized by these *cardinality tuples*. Using the path rules in $\mathcal{P}_{k^*}$, we will:

- efficiently sequence all admissible (i.e., path-realizable) cardinality tuples, and
- reconstruct the elements of $\mathcal{P}_{k^*}$ from those tuples.

For the algorithm, we can fix a canonical starting tuple that captures the maximal excursion which we will refer to often as the **seed tuple**. This seed tuple is analogous to what was referenced in Section 4 and defined formally in the path regime as Equation (4.2). With D and $k^* \geq 0$ fixed and $[0, k_+]$ as in Equation (5.1), we define this seed tuple $\hat{s}^{(0)}$, which is a valid $\hat{c}$ and is thus contained in $\mathcal{C}_{D,k^*}^+$, as the following element-wise:

$$(5.9) \quad c_k = \begin{cases} 0, & k_- \leq k < 0, \\ 1, & 0 \leq k \leq k^* - 1, \\ 2, & k^* \leq k \leq k_+ - 1, \quad (k \in [k_-, k_+]), \\ \begin{cases} 1, & D - k^* \text{ even,} \\ 2, & D - k^* \text{ odd,} \end{cases} & k = k_+, \end{cases}$$

Equivalently, $(c_k)_{k=k_-}^{k_+}$ has 0 for $k < 0$, then 1 on $[0, k^*)$, 2 on $[k^*, k_+)$, and a top entry at k_+ determined by the parity of $D - k^*$; this is the “highest” tuple corresponding to Equation (4.2), and $\sum_{k=k_-}^{k_+} c_k = D + 1$ by Equation (5.5). It serves as the starting point for the mass-shift enumeration. By Equation (5.6) and Equation (5.7), iterating over admissible tuples enumerates all equivalence classes and—via Equation (5.3)–Equation (5.8)—recovers the entire tree without redundancy and can be thought of, given the ordering scheme, as walking down the unique path representatives in the tree. Note that this maximal *tuple* lies in $\mathcal{C}_{D,k^*}^+$. We will later extend the enumeration from $\mathcal{C}_{D,k^*}^+$ to the full $\mathcal{C}_{D,k^*}$ (thus including $\mathcal{C}_{D,k^*}^-$) via an augmentation introduced after the construction. For now, as in Theorem 5.1, it is useful to begin with the maximal strictly positive case (including $k^* = 0$).

Remark 5.4. Before the enumeration process begins, it is important to note a structural constraint that follows directly from the graph. At all times during the redistribution procedure, the following conditions must hold:

1. For every index $0 \leq k \leq k^* - 1$, where we are considering arbitrary $\hat{c} \in \mathcal{C}_{D,k^*}^+$, we must maintain $c_k \geq 1$. This reflects the fact that each such slot must retain at least one visit a node in order to allow a path to ascend from 0 up to k^* .
2. For every index $k^* \leq k \leq k_+ - 1$, the mass in position c_k can only be decremented below 2 once the mass in the position immediately to its right,

c_{k+1} , has already been reduced to 0. This expresses the requirement that any excursion reaching level $k \geq k^*$ must eventually return downwards to k^* , which necessitates having at least two visits at those intermediate heights.

The only exception is the current highest occupied slot c_k at the peak of an excursion: this slot may equal 1, since its unit of mass can be redistributed while still preserving the condition that all admissible paths terminate at k^* . This graphical constraint also imposes a natural stopping condition to the algorithm, which we will discuss in detail later.

These constraints in [Theorem 5.4](#) are immediate from the structure of the underlying recombining tree and can be seen in a clear example where the red dotted line - which starts at k^* - requires that all elements below it (highlighted in yellow) meet condition (1) in [Theorem 5.4](#) and all elements at or above it meet condition (2) in [Theorem 5.4](#):

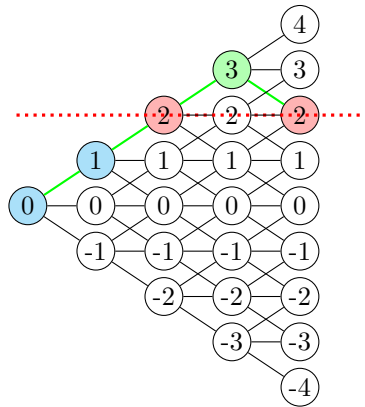


Fig. 4: Example of Graphically Informed Constraints on c_k

These constraints - informed by the structure of the graph - ensure that every admissible redistribution corresponds to a valid path ending at k^* while also remaining true to the minimality of the equivalence class representation of the tree [Equation \(5.8\)](#).

Working Example. Let's work through a simple example, taking $D = 7$ and $k^* = 2$. Using [Equation \(5.1\)](#), we have

$$k_- = -2 \quad \text{and} \quad k_+ = 4.$$

From [Equation \(4.2\)](#), the highest path with this $(k^*, D) = (2, 7)$ has position sequence

$$(5.10) \quad (0, 1, 2, 3, 4, 4, 3, 2)$$

and this has cardinality tuple $\hat{c}$:

$$(5.11) \quad \begin{matrix} -2 & -1 & 0 & 1 & 2 & 3 & 4 \\ (0, & 0, & 1, & 1, & 2, & 2, & 2) \end{matrix}$$

In line with Equation (5.5), we have

$$0 + 0 + 1 + 1 + 2 + 2 + 2 = 8.$$

Note the structure of Equation (5.11) compared to Equation (5.10). The path goes up to $k^* = 2$, then further up to 4, and then back down to $k^* = 2$. It visits 0 and 1 once (on the way up), and visits 2 and 3 twice (once up, once down). It also visits 4 twice (since $D - k^* = 5$ is odd), though it would visit 4 once if $D - k^*$ were even. In Equation (5.10) and Equation (5.11), the “way up” is in blue, the top of the excursion above $k^* = 2$ in green, and the remaining descent above $k^* = 2$ in red, similar in spirit to Figure 4.

Let’s secondly consider the next highest path reaching $(k^*, D) = (2, 7)$ which has cardinality tuple $\hat{c}$:

$$(5.12) \quad \begin{array}{ccccccc} -2 & -1 & 0 & 1 & 2 & 3 & 4 \\ (0, & 0, & 1, & 1, & 2, & 3, & 1) \end{array}$$

These are the paths $\mathbf{p}$:

$$(0, 1, 2, 3, 3, 4, 3, 2) \quad \text{and} \quad (0, 1, 2, 3, 4, 3, 3, 2).$$

In terms of the lexicographical ordering of Equation (4.1),

$$(0, 1, 2, 3, 4, 4, 3, 2) \succ_{\text{lex}} (0, 1, 2, 3, 4, 3, 3, 2) \succ_{\text{lex}} (0, 1, 2, 3, 3, 4, 3, 2).$$

In other words, the highest path Equation (5.10) is greater than the paths corresponding to Equation (5.12).

What is especially nice is that this same lexicographical ordering is maintained in the context of the cardinality tuples themselves. Here we compare tuples $\hat{c}$ right to left** (decreasing k), so the $\hat{c} = (0, 0, 1, 1, 2, 2, 2)$ is read as $\hat{c} = (2, 2, 2, 1, 1, 0, 0)$ for the comparison. Consequently, we can compare Equation (5.11) with Equation (5.12) as

$$(5.13) \quad \begin{array}{cccccccc} 4 & 3 & 2 & 1 & 0 & -1 & -2 & \\ (2, & 2, & 2, & 1, & 1, & 0, & 0) & \succ_{\text{lex}} (1, & 3, & 2, & 1, & 1, & 0, & 0) \end{array}$$

Consequently, the induced ordering between equivalence classes is determined by their cardinality tuples (under this right-to-left lex order we observed in Equation (4.1) and Theorem 5.3) and is independent of which path representative from each class is chosen. Here, we have pushed one of the parts of the top of the excursion above $k^* = 2$ (at height 4) back down into the lower parts of the excursion.

Let’s now examine the third highest path reaching $(k^*, D) = (2, 7)$ with cardinality tuple

$$(5.14) \quad \begin{array}{ccccccc} -2 & -1 & 0 & 1 & 2 & 3 & 4 \\ (0, & 0, & 1, & 1, & 3, & 2, & 1) \end{array}$$

These correspond to the paths:

$$(0, 1, 2, 2, 3, 4, 3, 2) \quad \text{and} \quad (0, 1, 2, 3, 4, 3, 2, 2).$$

There are exactly 2 paths corresponding to Equation (5.14). In terms of the lexicographical ordering of Equation (4.1), we therefore have

$$(0, 1, 2, 3, 4, 4, 3, 2) \succ_{\text{lex}} (0, 1, 2, 3, 4, 3, 3, 2) \succ_{\text{lex}} (0, 1, 2, 3, 4, 3, 2, 2) \succ_{\text{lex}} (0, 1, 2, 3, 3, 4, 3, 2) \succ_{\text{lex}} (0, 1, 2, 2, 3, 4, 3, 2).$$

In particular, the lower of the two paths from Equation (5.12), namely the path $(0, 1, 2, 3, 3, 4, 3, 2)$, still lies above both paths from Equation (5.14) except the path $(0, 1, 2, 3, 4, 3, 2, 2)$, which lies strictly between the two paths from Equation (5.12).

For the corresponding *cardinality tuples*, we compare entries from right to left (decreasing k) as before. Writing tuples in the order $4, 3, 2, 1, 0, -1, -2$, we have

$$\begin{array}{cccccccc} 4 & 3 & 2 & 1 & 0 & -1 & -2 & \\ (1, & 3, & 2, & 1, & 1, & 0, & 0) & \succ_{\text{lex}} (1, & 2, & 3, & 1, & 1, & 0, & 0) \end{array}$$

so the class for Equation (5.12) precedes the class for Equation (5.14) under this right-to-left lex order. As before, the induced ordering between equivalence classes is determined by their tuples and is independent of which path representative from each class is chosen. Here, compared with Equation (5.12), we have pushed *one* visit from level 3 down to level 2 (the single visit at level 4 remains unchanged).

Remark 5.5 (Truncating $\hat{c}$). We also reinforce the fact that we have clearly restricted attention to the positive set $\mathcal{C}_{D,k^*}^+$ as shown in Theorem 5.1 - which we will later augment to the full $\mathcal{C}_{D,k^*}$. Since every $\hat{c} \in \mathcal{C}_{D,k^*}^+$ has $c_k = 0$ for all $k < 0$, it is convenient to work with the truncated (positive-restriction) map

$$(5.15) \quad \text{Tr} : \mathcal{C}_{D,k^*}^+ \longrightarrow \text{Tr}(\mathcal{C}_{D,k^*}^+) \subseteq \mathbb{N}^{[0,k_+]}, \quad \text{Tr}((c_k)_{k_-}^{k_+}) \stackrel{\text{def}}{=} (c_k)_{k=0}^{k_+},$$

and we denote by Tr^{-1} its inverse on this image:

$$(5.16) \quad \text{Tr}^{-1} : \text{Tr}(\mathcal{C}_{D,k^*}^+) \longrightarrow \mathcal{C}_{D,k^*}^+, \quad \text{Tr}^{-1}((c_k)_0^{k_+}) \stackrel{\text{def}}{=} (\underbrace{0, \dots, 0}_{k=k_-, \dots, -1}, c_0, \dots, c_{k_+}).$$

On $\mathcal{C}_{D,k^*}^+$ we have $\text{Tr}^{-1} \circ \text{Tr} = \text{id}_{\mathcal{C}_{D,k^*}^+}$ and $\text{Tr} \circ \text{Tr}^{-1} = \text{id}_{\text{Tr}(\mathcal{C}_{D,k^*}^+)}$, so no information is lost by dropping the identically zero negative coordinates. The right-to-left lexicographic order is also preserved, because the discarded entries are equal (all zeros) for every element of $\mathcal{C}_{D,k^*}^+$.

Returning to the running example with $D = 7$, $k^* = 2$, $k_- = -2$, $k_+ = 4$, we see that the length-7 tuples in $\mathcal{C}_{7,2}^+$ (indexed by $-2, -1, 0, 1, 2, 3, 4$) correspond bijectively (c.f. Theorem 5.5) to length-5 truncated tuples (indexed by $0, 1, 2, 3, 4$):

$$\begin{aligned} (0, 0, 1, 1, 2, 2, 2) &\longleftrightarrow (1, 1, 2, 2, 2), \\ (0, 0, 1, 1, 2, 3, 1) &\longleftrightarrow (1, 1, 2, 3, 1), \\ (0, 0, 1, 1, 3, 2, 1) &\longleftrightarrow (1, 1, 3, 2, 1), \\ (0, 0, 1, 2, 2, 2, 1) &\longleftrightarrow (1, 2, 2, 2, 1), \\ (0, 0, 2, 1, 2, 2, 1) &\longleftrightarrow (2, 1, 2, 2, 1). \end{aligned}$$

where we completed this procedure with the last two entries into the list. Thus we notice that this procedure is actually the movement of mass $i = 1$ through the tuple.

The “move one unit of mass across the nonzero support” visualization can be carried out on the truncated tuples as an addition:

$$\begin{aligned}
 (5.17) \quad & (1, 1, 2, 2, 1) + (0, 0, 0, 0, 1) = (1, 1, 2, 2, 2) \\
 & \implies (1, 1, 2, 2, 1) + (0, 0, 0, 1, 0) = (1, 1, 2, 3, 1) \\
 & \implies (1, 1, 2, 2, 1) + (0, 0, 1, 0, 0) = (1, 1, 3, 2, 1) \\
 & \implies (1, 1, 2, 2, 1) + (0, 1, 0, 0, 0) = (1, 2, 2, 2, 1) \\
 & \implies (1, 1, 2, 2, 1) + (1, 0, 0, 0, 0) = (2, 1, 2, 2, 1)
 \end{aligned}$$

Note that whenever needed, we recover the full length-7 representatives in $\mathcal{C}_{7,2}^+$ by applying Tr^{-1} , i.e., by padding two leading zeros (the entries for $k = -2, -1$). In particular, the truncated tuple $(1, 1, 2, 3, 1)$ is exactly the positive-restriction of the length-7 tuple $(0, 0, 1, 1, 2, 3, 1)$, and similarly for the others. Hence all ordering and enumeration arguments carry over verbatim on $\mathcal{C}_{D,k^*}^+$ using truncated tuples, with no loss of correctness or generality.

This same process in Equation (5.17) can be captured by graphically traversing the tree. To do this, we consider a process where a unit of mass $i = 1$ is subtracted from the final entry of $\hat{c}$, and this mass is “moved” leftward through the tuple, one position at a time. This results in the following sequence of configurations which we can interpret as a walk down the unique path representatives in the graph:

(5.18)

$(0, 0, \dots, 0, 1)$
 $(0, 0, \dots, 1, 0)$
 $\vdots$
 $(0, 1, \dots, 0, 0)$
 $(1, 0, \dots, 0, 0)$

$\implies$

Sample Path Configuration

$(1, 1, 1, 1, 1, 2)$	$(1, 1, 1, 1, 2, 1)$	$(1, 1, 1, 2, 1, 1)$
$(0, 0, 0, 0, 0, 1)$	$(0, 0, 0, 0, 1, 0)$	$(0, 0, 0, 1, 0, 0)$
$(1, 1, 2, 1, 1, 1)$	$(1, 2, 1, 1, 1, 1)$	$(2, 1, 1, 1, 1, 1)$
$(0, 0, 1, 0, 0, 0)$	$(0, 1, 0, 0, 0, 0)$	$(1, 0, 0, 0, 0, 0)$

As can be seen in Equation (5.18), the “one-mass sweep” already exhausts all unique configurations when $k^* = D - 1$: by the equivalence relation Equation (5.7) and the graphical observations in Equation (5.18), shifting zero mass (the starting tuple Equation (5.9)) and then one unit from the rightmost entry across the support generates every admissible class at that depth. For deeper targets $k^* < D - 1$ (e.g., the toy case $D = 7$, $k^* = 2$), additional configurations appear and we must move larger amounts of mass while preserving feasibility of paths (hence of the cardinality tuples) to generate **all** the possible configurations.

To organize this, we define the number of *available slots*, for $\hat{c} \in \mathcal{C}_{D,k^*}^+$ at a given stage to be the count of indices in the truncated support $[0, k_+]$ that a unit of mass may traverse from the current rightmost non-zero index:

$$\ell \stackrel{\text{def}}{=} k_+ - 0 + 1 = k_+ + 1.$$

In the toy example, $\ell = 5$. A unit of mass may be removed only from the final element c_{k_+} and shifted left through these ℓ slots; this produces exactly the one-mass family.

Once c_{k+} is depleted, we proceed to c_{k+-1} , and ℓ decreases by one (we never “move over zero slots,” which would duplicate earlier tuples or violate feasibility). To reach new configurations, we then have two units we need to move, and so on: at stage m we permit moving m units while the available slots shrink as the right boundary retreats. This schedule respects the graph constraints at every step and, under the lexicographic order, enumerates all admissible tuples without omission or repetition.

Returning to our toy example, we remove two units of mass from the rightmost entry of an arbitrary seed $\hat{c} \in \mathcal{C}_{7,2}^+$ whose elements are described by Equation (5.9). The construction proceeds analogously to Equation (5.17): the mass is first subtracted, then permuted across the ℓ available slots, and finally these permutations are added back to the originating tuple. This additive perspective serves only to illustrate the mechanism; the algorithm itself does not require explicitly performing these additions.

Now that we are at stage $m = 2$, we need to move $i = m = 2$ mass across $\ell = 4$ many slots. One would naturally ask the question, what about $(0, 0, 0, 0, 2) + (1, 1, 2, 2, 0)$? Our algorithm naturally captures this through our seed tuple $\hat{s}^{(0)}$ Equation (5.9), so we don’t want to count it again. But, by construction of the algorithm, we automatically exclude this case from being counted by how we start this second stage i.e.:

$$(5.19) \quad \left(\boxed{2}, 1, 2, \boxed{2}, \boxed{1} \right) \Rightarrow (1, 1, 2, 4, 0)$$

Remark 5.6. Note as well that this same process happened bridging our mass $m = i = 0$ stage to our $m = i = 1$ stage which built in this natural exclusion of the seed tuple $\hat{s}^{(0)}$ from the $i = 1$ case as well:

$$(5.20) \quad \left(\boxed{1}, 1, 2, \boxed{2}, \boxed{2} \right) \Rightarrow (1, 1, 2, 3, 1)$$

In this case we removed 0 mass from the leftmost entry (since taking more would violate the tree structure) and 1 mass from the rightmost entry (recovering the seed configuration). Thus we moved $i = 1$ across $\ell = 4$ slots and $i = 0$ across $\ell = 5$ slots, exactly as predicted. Summing these possibilities gives 5 distinct combinations, in agreement with Equation (5.17).

By construction, this procedure never under- or over-counts equivalence classes, preserving the minimality of the relation. Moreover, the same mechanism applies at every stage: the slot-mass combinatorics automatically encode the correct enumeration. With these tools equipped, we can finally enumerate the $m = i = 2$ stage for

our toy example:

$$\begin{aligned}
 & (1, 1, 2, 2, 0) + (0, 0, 0, 2, 0) = (1, 1, 2, 4, 0) \\
 \implies & (1, 1, 2, 2, 0) + (0, 0, 1, 1, 0) = (1, 1, 3, 3, 0) \\
 \implies & (1, 1, 2, 2, 0) + (0, 1, 0, 1, 0) = (1, 2, 2, 3, 0) \\
 \implies & (1, 1, 2, 2, 0) + (1, 0, 0, 1, 0) = (2, 1, 2, 3, 0) \\
 \implies & (1, 1, 2, 2, 0) + (0, 0, 2, 0, 0) = (1, 1, 4, 2, 0) \\
 \implies & (1, 1, 2, 2, 0) + (0, 1, 1, 0, 0) = (1, 2, 3, 2, 0) \\
 \implies & (1, 1, 2, 2, 0) + (1, 0, 1, 0, 0) = (2, 1, 3, 2, 0) \\
 \implies & (1, 1, 2, 2, 0) + (0, 2, 0, 0, 0) = (1, 3, 2, 2, 0) \\
 \implies & (1, 1, 2, 2, 0) + (1, 1, 0, 0, 0) = (2, 2, 2, 2, 0) \\
 \implies & (1, 1, 2, 2, 0) + (2, 0, 0, 0, 0) = (3, 1, 2, 2, 0)
 \end{aligned}
 \tag{5.21}$$

What [Equation \(5.17\)](#) and [Equation \(5.21\)](#) show is that we are effectively counting combinations of integer partitions over a truncated tuple of size ℓ , and then reinserting that truncated structure into the stage- m starting tuple (cf. [Equation \(5.19\)](#), [Equation \(5.20\)](#)). In particular, comparing [Equation \(5.17\)](#) and [Equation \(5.21\)](#) with the integer partitions of $i = 1, 2$ demonstrates that this identification is valid without loss of generality, accuracy, or minimality of the equivalence relation:

- For $i = 1$, there is only one partition: (1) .
- For $i = 2$, there are two partitions: (2) and $(1 + 1)$.

At any stage, the validity of this procedure can be verified empirically by examining the tree graph for fixed k^* and D (both finite for case studies):

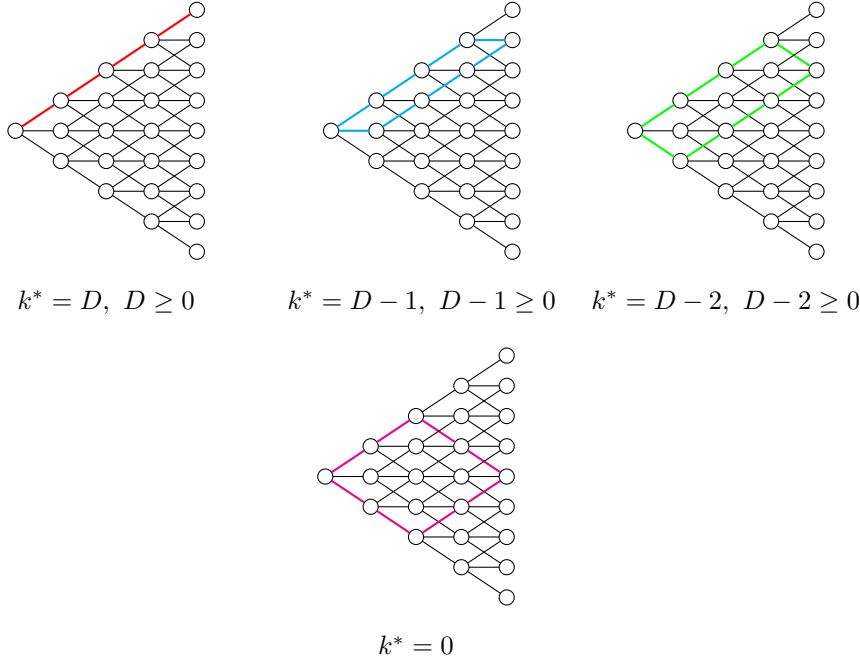


Fig. 5: Recombining Tree Paths

This graphical viewpoint, together with the toy example and algorithm, provides the foundation for developing closed-form combinatorics. We develop said combinatorics by introducing the well-known formula for counting "weak compositions" [20]:

$$(5.22) \quad \mathcal{C}^w = \binom{i + \ell - 1}{\ell - 1}$$

This formula gives an exact count of the ways to distribute a mass i across ℓ slots and, in doing so, establishes a precise correspondence between the enumeration of the equivalence classes of tree structures and the theory of integer partitions - captured formally by the weak composition formula. Thus, this formula can be exploited to obtain a closed-form combinatorial expression with a corresponding algorithm that enables one to either (a) count and enumerate **exactly** the number of **unique** paths reaching a node k^* of their choice in a recombining trinomial tree of depth D , or (b) to generate **all** unique paths in a recombining trinomial tree of depth D by iterating over all $k^* \in [-D, D]$.

In order to achieve this, we must take into account the dynamic nature of the size of the mass, the slots available to us during each stage, and finally, the existence of negative paths and how we can extend these observations and the combinatorial expression to capture the entire general structure (i.e. the case $k^* \in \{D, D - 1, \dots, 0, \dots, -D + 1, -D\}$). In order to achieve this, we perform a case study, which we will be able to link to our toy example, and then continuously extend it

until we have generalized to this case.

5.1.1. Case Examination:

Case 1: $k^* = D$, $D \geq 0$. In Equation (5.9) we introduced a seed tuple as the general template for initialization; in the toy example this specialized to Equation (5.11). One perspective on this seeding is provided by the weak composition formula: here we move $m = i = 0$ units of mass across a tuple with $\ell = 5$ available slots,

$$\binom{0+5-1}{5-1} = \binom{4}{4} = 1,$$

confirming that the initialization consists of a single cardinality tuple. More generally, when $k^* = D$ the same phenomenon occurs (cf. Figure 5): the algorithm is seeded, no mass can be moved without violating the tuple (and hence the tree), and exactly one maximal path results.

Formally,

$$(5.23) \quad \mathcal{C}_{D, k^*=D}^w = \binom{0+\ell-1}{\ell-1} = \binom{\ell-1}{\ell-1} = 1.$$

In this case, the set $\mathcal{C}_{D, k^*=D}$ contains a single cardinality tuple, so $|\mathcal{C}_{D, k^*=D}| = 1$. The weak-composition procedure $\mathcal{C}_{D, k^*=D}^w$ therefore enumerates exactly one configuration, establishing $|\mathcal{C}_{D, k^*=D}^w| = |\mathcal{C}_{D, k^*=D}| = 1$. In our toy example $k^* \neq D$, so further stages are required beyond this trivial case, and we therefore proceed to Case 2.

Case 2: $k^* = D - 1$, $D - 1 \geq 0$. In this case we recover exactly the situation of our worked example in Equation (5.18): the process reduces to moving a single unit of mass across the cardinality tuple, which corresponds to stepping down one level at a time in the graph in order to generate the next collection of possible paths, naturally following the lexicographical ordering. This procedure generates all unique legal paths, and—by manipulating the tuple entries—can equivalently be used to reconstruct all corresponding tuples. Moreover, as illustrated in Figure 5, no negative excursions are possible at this depth, so no augmentation of the counting process is required.

Formally, the enumeration reduces to the positive set, $\mathcal{C}_{D, k^*=D-1}^+ = \mathcal{C}_{D, k^*=D-1}$. We again apply the weak composition formulation, verified directly in Equation (5.17). The seed tuple $\hat{s}^{(0)}$ (the first tuple enumerating in Equation (5.17)) is already counted by $\mathcal{C}_{D, k^*=D}^w$, so it remains only to count the ways of moving $m = i = 1$ unit of mass through $\ell - 1$ slots and add this to the base case. Thus,

$$(5.24) \quad \mathcal{C}_{D, k^*=D-1}^w = \mathcal{C}_{D, k^*=D}^w + \binom{1+(\ell-1)-1}{(\ell-1)-1} = 1 + \binom{\ell-1}{\ell-2} = \ell.$$

This agrees with the explicit enumeration in Equation (5.17) (where $\ell = 5$ gives exactly 5 paths and we enumerated 5 paths) and matches the walk down the path in the graphical structure of Figure 5.

Recognizing the Algorithm. For $m = i = 2$ an algorithmic pattern emerges. The seed tuple $\hat{s}^{(0)}$ (cf. Equation (5.9)) has its entries determined by the depth D and the terminal index k^* . By parity, the last entry c_{k^*} is either 1 or 2, so the mass-redistribution count must branch on this value. This is already visible in Case 2 (5.1.1): moving one unit requires counting over $\ell - 1$ slots, in addition to the trivial $i = 0$ move over ℓ slots. In our toy example $\mathcal{C}_{D=7, k^*=2}$, the final entry

begins at 2, so this issue does not arise immediately; however, if $c_{k_+} = 1$ (which is permitted and exhibited by Equation (5.9)), that step exhausts the available mass at the boundary, and to avoid repeats (and to continue the lexicographically ordered “march down” the tree) we must proceed with two units moved over $\ell - 2$ slots instead. In short, the enumeration at each stage respects the lexicographic schedule and the tree’s feasibility constraints, but its first increment depends on the parity of the terminal entry. Accordingly, we introduce the switching term

$$(5.25) \quad \beta = \begin{cases} 1, & D + k^* \equiv 1 \pmod{2}, \\ 2, & D + k^* \equiv 0 \pmod{2}, \end{cases}$$

which dictates whether the initial increment at the right boundary consumes one or two units before the process resumes its lexicographic descent. If this term’s use is not apparent now, as we write the full combinatorics, its use will become obvious.

With the switching term β in place, we separate the counting process into two regimes: the *even* and *odd* cases. The need for this distinction comes from the observation that the number of available slots ℓ depends on the parity of the terminal entry c_{k_+} in the seed tuple. This parity directly alters the redistribution schedule and hence modifies the weak composition count $\mathcal{C}^w$. Consequently, in the general case the formula itself must adapt, and β serves as the toggle between the two regimes.

We therefore write the weak composition enumeration of $\mathcal{C}_{D,k^*}^+$ for the m -th stage, valid for any $m > 0$ and any $i \geq 0$, in two versions: one for the even case ($\beta = 2$) and one for the odd case ($\beta = 1$). This separation ensures that the lexicographic march down the tree is preserved and that no paths are over- or under-counted.

Odd Case: $c_{k_+} = 1, \beta = 1$

stage (m)	mass (i)	slots (ℓ)	#compositions (C^w)
0	0	ℓ	1
1	1	$\ell - 1$	$\binom{i+\ell-2}{\ell-2}$
2	2	$\ell - 2$	$\binom{i+\ell-3}{\ell-3}$
3	3	$\ell - 2$	$\binom{i+\ell-3}{\ell-3}$
4	4	$\ell - 3$	$\binom{i+\ell-4}{\ell-4}$
5	5	$\ell - 3$	$\binom{i+\ell-4}{\ell-4}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	$i = D - k^*$	$\ell - \lceil \frac{i+1}{2} \rceil, i > 0$	$\binom{i+\ell-\lceil \frac{i+1}{2} \rceil-1}{\ell-\lceil \frac{i+1}{2} \rceil-1}$

Even Case: $c_{k_+} = 2, \beta = 2$

stage (m)	mass (i)	slots (ℓ)	#compositions (C^w)
0	0	ℓ	1
1	1	$\ell - 1$	$\binom{i+\ell-2}{\ell-2}$
2	2	$\ell - 1$	$\binom{i+\ell-2}{\ell-2}$
3	3	$\ell - 2$	$\binom{i+\ell-3}{\ell-3}$
4	4	$\ell - 2$	$\binom{i+\ell-3}{\ell-3}$
5	5	$\ell - 3$	$\binom{i+\ell-4}{\ell-4}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	$i = D - k^*$	$\ell - \lfloor \frac{i+1}{2} \rfloor$	$\binom{i+\ell-\lfloor \frac{i+1}{2} \rfloor-1}{\ell-\lfloor \frac{i+1}{2} \rfloor-1}$

Table 1: Weak Composition Patterns with Parity-Based Indexing

Returning to the toy example $\mathcal{C}_{7,2}$, we observe that $c_{k_+} = 2$, so $\beta = 2$ and we are in the even regime. Referring to the even-case table, we compare the stage $m = i = 2$ with the explicit enumeration in Equation (5.21). Substituting into the weak-composition formula gives

$$\binom{2+5-2}{5-2} = \binom{5}{3} = 10,$$

which matches the number of tuples explicitly listed.

To account for all positive paths up to this point, we note that $\mathcal{C}_{D,k^*=D-2}^w$ should include: mass 0 over $\ell = 5$, mass 1 over $\ell = 4$, and mass 2 over $\ell = 4$. This yields

$$\begin{aligned}
 \mathcal{C}_{D,k^*=D-2}^w &= \mathcal{C}_{D,k^*=D}^w + \mathcal{C}_{D,k^*=D-1}^w + \binom{2+(\ell-1)-1}{(\ell-1)-1} \\
 (5.26) \quad &= 1 + (\ell-1) + \binom{\ell}{\ell-2}.
 \end{aligned}$$

For $\ell = 5$, this gives 15 total combinations, exactly matching the enumerated tuples in Equation (5.21). However, careful inspection of the graph in Figure 5 reveals the

first appearance of a negative path. Since Equation (5.26) accounts only for positive paths, it under-counts by precisely one and is thus incomplete. This observation leads naturally to the next case, $k^* = D - 2$, $D - 2 \geq 0$, where negative contributions must begin to be incorporated.

Remark 5.7. At this point it is useful to note that all positive paths can be systematically counted within the even-odd regime using Table 1. In particular, expression Equation (5.26), though incorrect, highlights the general form of the positive-path count $\mathcal{C}_{D,k^*}^+$, which can use Table 1 to write. Again, defer the correct count in Equation (5.26), which has negative paths, to the subsequent cases, beginning with $k^* = D - 2$, $D - 2 \geq 0$, and continuing with the general case $k^* \in \{D, D - 1, \dots, 0, \dots, -D + 1, -D\}$, which will generalize and ultimately complete the counting procedure for any D, k^* . In order to write Table 1 into a nice compact closed-form non-piecewise combinatorial expression, we utilize the well known identity relating the ceiling and floor functions:

$$(5.27) \quad \left\lceil \frac{i+1}{2} \right\rceil = \left\lfloor \frac{i+1}{2} \right\rfloor + \delta, \quad \delta = \begin{cases} 1, & i \text{ is even} \\ 0, & i \text{ is odd} \end{cases}$$

We also recall Pascal's Identity:

$$(5.28) \quad \binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Examining the general case in Table 1, define

$$a \stackrel{\text{def}}{=} \left\lfloor \frac{i+1}{2} \right\rfloor, \quad c \stackrel{\text{def}}{=} \left\lceil \frac{i+1}{2} \right\rceil = a + \delta(i).$$

The “even-case” general term then reads

$$(5.29) \quad \binom{i+\ell-a-1}{\ell-a-1} = \binom{i+\ell-\lfloor \frac{i+1}{2} \rfloor - 1}{\ell - \lfloor \frac{i+1}{2} \rfloor - 1}.$$

Applying Equation (5.28) to Equation (5.29) yields

$$(5.30) \quad \binom{i+\ell-a-1}{\ell-a-1} = \binom{i+\ell-a-2}{\ell-a-1} + \binom{i+\ell-a-2}{\ell-a-2}.$$

Rewriting in terms of $c = a + \delta(i)$ gives the uniform identity

$$(5.31) \quad \binom{i+\ell-a-1}{\ell-a-1} = \binom{i+\ell-c-1}{\ell-c-1} + \delta(i) \binom{i+\ell-c-1}{\ell-c}.$$

Indeed, if i is odd then $\delta(i) = 0$ and the second term vanishes; if i is even then $\delta(i) = 1$ and Equation (5.31) is precisely the Pascal split with indices shifted by one. Next, we use the global parity switch determined by (D, k^*) that we introduced in Equation (5.25) so that $\beta - 1 \in \{0, 1\}$. In regimes where the parity choice is fixed by (D, k^*) (and not by i), we replace the local $\delta(i)$ in Equation (5.31) by the global toggle $\beta - 1$. Using $c = \lceil \frac{i+1}{2} \rceil$, this yields the single closed-form summand

$$(5.32) \quad \binom{i+\ell-c-1}{\ell-c-1} + (\beta - 1) \binom{i+\ell-c-1}{\ell-c}, \quad c = \left\lceil \frac{i+1}{2} \right\rceil.$$

Therefore, the positive-path count, which as we saw from Equation (5.26), can taken as then sum over the mass $i = 0$ until $D - k^*$. Given this, for any arbitrary starting tuple $\hat{c}$, given the choice of D, k^* can be written without case splits as

$$(5.33) \quad \mathcal{C}_{D,k^*}^{w,+}(\hat{c}) = \sum_{i=0}^{D-k^*} \left\{ \binom{i + \ell - \lceil \frac{i+1}{2} \rceil - 1}{\ell - \lceil \frac{i+1}{2} \rceil - 1} + (\beta - 1) \binom{i + \ell - \lceil \frac{i+1}{2} \rceil - 1}{\ell - \lceil \frac{i+1}{2} \rceil} \right\}$$

We write $\mathcal{C}_{D,k^*}^{w,+}$ to reconcile with Equation (5.26): here the superscript $w, +$ indicates the count of *weak-composition terms arising from positive paths only*, i.e., paths that never visit the negative side, thereby excluding the negative-path contribution. This distinguishes $\mathcal{C}_{D,k^*}^{w,+}$ from the full weak-composition count $\mathcal{C}_{D,k^*}^w$, which includes both positive and negative paths (the latter will be incorporated in the subsequent cases).

For the stopping condition of the counting process - and by extension the algorithm - we must make explicit the feasibility constraint on the cardinality tuple. Along the positive side, every level up to the terminal index must remain reachable; equivalently, for each level $\ell \leq k^*$ we must reserve the baseline occupancy that maintains reachability (otherwise some node becomes unreachable). In addition, the terminal level k^* contributes its fixed terminal mass.

It is important to note that k^* itself may tick down from its default value of 2 to 1, but never below 1 and values of $\hat{c}$, $c_{0 \leq k < k^*}$, cannot give up any of their mass thus they remain at 1 and do not contribute to the sum bounds. Allowing $k^* < 1$ would immediately force the path into an unreachable state and invalidate the composition. Thus, when constructing the admissible range for the transferable surplus mass i , we must account for this minimal cutoff. With the baseline and terminal reservations enforced, the maximum transferable mass is $i_{\max} = D - k^*$, since algebraically $i = D - k^* - 1 + 1 = D - k^*$. Therefore the admissible summation range is

$$(5.34) \quad 0 \leq i \leq D - k^*,$$

and this upper bound furnishes the stopping criterion used by the counting algorithm for $\mathcal{C}_{D,k^*}^{w,+}$.²

We can now finally proceed to our last two cases and in doing so, augment our combinatorics and algorithm with the proper introduction of negative paths. In order to do this, we need to return to observation we made in our initial construction of the tree: symmetry.

Symmetry of the Recombining Tree. One of the core advantages of the recombining tree is its symmetry over the path $\mathbf{p} = ((0, 0), (0, 0), \dots, (0, 0))$ as referenced in Section 2. In our construction of the problem, we continuously referenced positive paths, and then mixed paths that contained both positive and negative paths. Moreover, in Figure 5, we structured these cases over the positive paths. This property of symmetry allows us to state the follow theorem:

THEOREM 5.8 (Reflection symmetry of counts). *Let $\mathcal{T}_D = (V_D, E_D)$ be the*

²Additional Remarks:

- (i) If $D + k^*$ is odd, then $\beta = 1$ and the second binomial in each summand is suppressed.
- (ii) If $D + k^*$ is even, then $\beta = 2$ and one recovers the full Pascal contribution, matching the “even-regime” rows of Table 1.

recombining rooted trinomial tree with

$$\begin{aligned} V_D &= \{(k, d) \in \mathbb{Z} \times \mathbb{Z}_+ : 0 \leq d \leq D, |k| \leq d\} \\ E_D &= \{((k, d-1), (k+s, d)) : s \in \{-1, 0, 1\}\} \end{aligned}$$

Let $\mathcal{P}^+$ (resp. $\mathcal{P}^-$) be the set of root-to-depth- D paths that stay on the nonnegative (resp. non-positive) side, i.e. $k_d \geq 0$ (resp. $k_d \leq 0$) for all d . For any terminal constraint that is symmetric under $k \mapsto -k$ (e.g., “end at $\pm k^\dagger$ ” or “end in a set S with $S = -S$), the map

$$(5.35) \quad R : (k, d) \mapsto (-k, d)$$

induces a bijection $R : \mathcal{P}^+ \rightarrow \mathcal{P}^-$, and hence a bijection on cardinality tuples

$$(5.36) \quad \hat{C} : \pi \mapsto (c_k(\pi))_k \quad \text{satisfies} \quad \hat{C}(R\pi) = \rho(\hat{C}(\pi)), \quad \text{where } \rho((c_k)_k) = (c_{-k})_k.$$

Consequently, the positive- and negative-side combinatorial counts coincide:

$$\#\hat{C}(\mathcal{P}^+) = \#\hat{C}(\mathcal{P}^-),$$

and any enumeration algorithm (or closed-form count) developed on $\mathcal{P}^+$ applies mutatis mutandis to $\mathcal{P}^-$ via ρ .

Proof. Define $R(k, d) = (-k, d)$. Then R is a graph automorphism of $\mathcal{T}_D$: if $((k, d-1), (k+s, d)) \in E_D$ with $s \in \{-1, 0, 1\}$, applying R to both endpoints yields

$$((-k, d-1), (-k-s, d)),$$

which is again an edge in E_D since $-s \in \{-1, 0, 1\}$. The root $(0, 0)$ is fixed by R , and depth is preserved.

If $\pi \in \mathcal{P}^+$ satisfies a symmetric terminal constraint (e.g., $k_D \in S$ with $S = -S$ or $k_D = \pm k^\dagger$), then $R\pi \in \mathcal{P}^-$ satisfies the same constraint because R flips the sign of k and leaves d unchanged. In particular, R is a bijection $\mathcal{P}^+ \rightarrow \mathcal{P}^-$ with inverse R itself.

For the cardinality tuple $\hat{C}(\pi) = (c_k(\pi))_k$, where $c_k(\pi) = |\{d \in \{0, \dots, D\} : k_d = k\}|$, one has

$$c_k(R\pi) = |\{d : -k_d = k\}| = |\{d : k_d = -k\}| = c_{-k}(\pi),$$

so $\hat{C}(R\pi) = \rho(\hat{C}(\pi))$ with $\rho((c_k)_k) = (c_{-k})_k$. Thus ρ is a bijection between the sets of cardinality tuples realized by $\mathcal{P}^+$ and $\mathcal{P}^-$, which proves the equality of counts.

Finally, any enumeration algorithm on $\mathcal{P}^+$ that operates by (i) local transitions $k \mapsto k+s$ with $s \in \{-1, 0, 1\}$ and (ii) book-keeping via the tuple $(c_k)_k$ is equivariant under ρ ; hence reflecting the output (or equivalently mirroring indices $k \mapsto -k$ in the algorithm) yields a correct enumeration on $\mathcal{P}^-$. $\square$

Case 3: $k^* = D-2$, $D-2 \geq 0$. Like [Subsection 5.1.1](#), we can use the graphical structure to see what amounts to a trivial case where we add our first negative path. Examining [Figure 5](#) we can see that there is only 1 mixed positive and negative path. Thus, we can finally correct [Equation \(5.26\)](#) by counting the negative path formally:

$$\begin{aligned} \mathcal{C}_{D, k^*=D-2}^w &= \mathcal{C}_{D, k^*=D-2}^{w,-} + \mathcal{C}_{D, k^*=D-2}^{w,+} \\ (5.37) \quad &= 1 + \sum_{i=0}^2 \left\{ \binom{i+\ell - \lceil \frac{i+1}{2} \rceil - 1}{\ell - \lceil \frac{i+1}{2} \rceil - 1} + (\beta-1) \binom{i+\ell - \lceil \frac{i+1}{2} \rceil - 1}{\ell - \lceil \frac{i+1}{2} \rceil} \right\} \end{aligned}$$

Plugging in the correct β, ℓ for our toy example $\mathcal{C}_{7,2}$ (where we can finally dispense with the superscript $+$ notation), verifies that this count is 16, exactly what we would expect.

Unfortunately, while it is possible, it now becomes difficult to see in the graph all of the negative paths and their combinatorial structure. However, using [Theorem 5.8](#), we know that our counting process, so long as we follow the constraints, will remain valid. This observation however is key and leads to some key questions. 1) How can we maintain the fidelity of the algorithm while still optimizing for computational speed? 2) What constraints does the tree impose that we need to account for to maintain the equivalence relation and still be able to utilize [Equation \(5.33\)](#)? 3) How can we exploit our original and canonical construction $\hat{C}(\mathbf{p})$? Fortunately, in our next and final case, we will not only be able to answer all these questions, but also be able to construct our full algorithm and complete combinatorial expression.

Case N: $k^* \in \{-D, -D+1, \dots, D\}$. Take any mixed-integer path $\mathbf{p} \in \mathcal{P}_{D,k^*}$ that satisfies the standing constraints of our construction: successive nodes differ by ± 1 or 0, the path starts at 0, and it must terminate at the prescribed endpoint k^* . For paths that extend into the negative region, however, an additional restriction arises, one that is also exploited in [Section E](#) when we wrote our recursiveless enumeration in [Section 4](#). Specifically, a valid traversal requires the return to the nonnegative positions. Concretely, whenever our mixed $\mathbf{p}$ takes its step -1 (or more negative pieces subject to obeying the same constraints) while considering $k^* \geq 0$, at least one additional 0-step becomes necessary to ensure a return to the positive side:

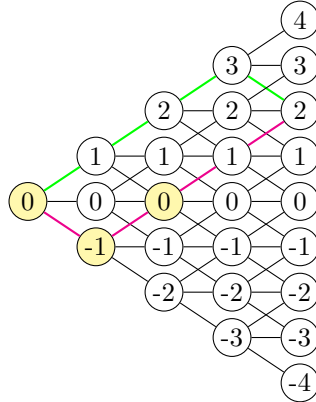


Fig. 6: Mixed-integer recombining tree highlighting a sample traversal for $k^* = D - 2$ (green).

Here, we highlight an example of the negative path in magenta and the use of the additional required 0 when a negative value is hit. Further inspection will show that whenever a negative node is traversed in a valid path $\mathbf{p}$, at least two visits to 0 are required. By symmetry, the same holds whenever $k^* < 0$. In the special case $k^* = 0$, three distinct visits to 0 are necessary: the initial step, the return, and the terminal position.

Accordingly, the admissible cardinality tuples must satisfy

$$(5.38) \quad c_0 = \begin{cases} 2, & k^* \neq 0, \\ 3, & k^* = 0. \end{cases}$$

In our definition [Equation \(5.9\)](#), we prescribed minimum counts for certain c_k values that must be preserved throughout the mass-redistribution process. Specifically, entries initialized at 1 remain fixed, while entries initialized at 2 may be reduced provided they are walked down appropriately through the redistribution procedure described in the previous cases. The new condition augments this rule: whenever a path includes a negative node, the process terminates precisely when mass must first be removed from k^* , at which point c_{k^*} is reduced to 1. This stopping condition is exactly the one encoded in the upper bound of the summation in [Equation \(5.34\)](#) and keeps us from ever decreasing c_0 below 2.

In the same vein, the negative paths obey analogous constraints to the positive ones. As the tree is traversed into negative indices, the cardinality counts must satisfy nested lower bounds. Assume $k_- < 0$ and let $m_* \stackrel{\text{def}}{=} -k_- \in \mathbb{Z}_{>0}$. To admit excursions down to k_- , the cardinalities must satisfy

$$(5.39) \quad c_{k_-} \geq 1, \quad c_{-j} \geq 2 \quad \text{for } 1 \leq j \leq m_* - 1,$$

together with c_0 as in [Equation \(5.38\)](#).

Equivalently, for each intermediate depth $m \in \{1, \dots, m_*\}$, inclusion of the node $-m$ requires

$$(5.40) \quad c_{-j} \geq \begin{cases} 2, & 1 \leq j < m, \\ 1, & j = m, \end{cases} \quad (1 \leq j \leq m).$$

With these new constraints equipped, we can use our understanding of our lexicographical ordering to inform how, while obeying these constraints, we can enumerate all these paths. Previously, when generating all positive paths, our process naturally maintained the lexicographical ordering and thus verified our enumeration algorithm was not missing any paths and our combinatorics were not missing any counts. We can impose this same logic on the negative paths by extending our lexicographical ordering for the cardinality tuples. We state it as such:

Remark 5.9 (Extension of lexicographic order to mixed-sign tuples). Let $\hat{c} = (c_k)_{k=k_-}^{k_+}$ and $\hat{c}' = (c'_k)_{k=k_-}^{k_+}$ be elements of $\mathcal{C}_{D,k^*} \subset \mathbb{N}^{[k_-, k_+]}$. We define the index blocks of c_k

$$(5.41) \quad I_- \stackrel{\text{def}}{=} (-1, -2, \dots, k_-), \quad I_+ \stackrel{\text{def}}{=} (k_+, k_+ - 1, \dots, 0),$$

and define the comparison key:

$$(5.42) \quad \Phi(\hat{c}) \stackrel{\text{def}}{=} (-c_{-1}, -c_{-2}, \dots, -c_{k_-}; c_{k_+}, c_{k_+ - 1}, \dots, c_0) \in \mathbb{Z}^{|I_-| + |I_+|}.$$

Equip $\mathbb{Z}^{|I_-| + |I_+|}$ with the standard (left-to-right) lexicographic order. We then define the *negative-first, right-to-left* lex order on $\mathcal{C}_{D,k^*}$ by

$$(5.43) \quad \hat{c} \prec_{\text{lex} \pm} \hat{c}' \iff \Phi(\hat{c}) \text{ is lexicographically smaller than } \Phi(\hat{c}').$$

Unpacking the rule:

1. (*Compare the negative part first; “smaller is larger” deeper down.*) Let $j^* = \min\{j \geq 1 : c_{-j} \neq c'_{-j}\}$, if it exists. Then

$$(5.44) \quad \hat{c} \succ_{\text{lex} \pm} \hat{c}' \iff c_{-j^*} < c'_{-j^*}.$$

For instance, $(c_{-1}, c_{-2}) = (2, 1)$ compares as $(2, 1) \succ_{\text{lex} \pm} (2, 2)$, mirroring $-21 > -22$.

2. (*Tie on negatives $\Rightarrow$ compare positives right-to-left in the usual sense.*) If $c_{-j} = c'_{-j}$ for all $j \geq 1$ with $-j \geq k_-$, set

$$(5.45) \quad i^* = \max\{i \in [0, k_+] : c_i \neq c'_i\}.$$

Then we have

$$(5.46) \quad \hat{c} \succ_{\text{lex} \pm} \hat{c}' \iff c_{i^*} > c'_{i^*}.$$

This order $\prec_{\text{lex} \pm}$ is a total order on $\mathcal{C}_{D,k^*}$; it *extends* the right-to-left lex order on $\mathcal{C}_{D,k^*}^+$ (where all negative coordinates are zero), and via the cardinality map Equation (5.3)–Equation (5.8) it induces a representative-independent total order on the equivalence classes in $\mathcal{P}_{k^*}$ (cf. Equation (4.1)). Moreover, it coincides perfectly with a walk down the mixed paths given by the graph of $k^* \in \{-D, -D+1, \dots, D\}$ where the number of negative paths that reach k^* is nontrivial. By reflection symmetry (Theorem 5.8), the strictly negative side is obtained by the involution $k \mapsto -k$, consistent with the sign flip built into Φ .

We now employ the extended lexicographical ordering together with the additional cardinality constraints and the combinatorial expression previously developed. Taken together, these three ingredients enable us to formulate the refined algorithm in its complete form and to derive the associated closed-form combinatorial expression. The key step is the construction of an appropriate *shift function*, which expands the enumeration while respecting the constraints - and of course is motivated by the ordering.

The Shift Function. Recall the positive maximal seed $\hat{s}^{(0)}$ from Equation (5.9). Our extended lexicographic order (Theorem 5.9) tells us to prioritize comparisons by the negative coordinates and only then, in case of ties, by the positive coordinates (right-to-left). To extend the enumeration from $\mathcal{C}_{D,k^*}^+$ to the full $\mathcal{C}_{D,k^*}$ while respecting Equation (5.38)–Equation (5.40), we proceed in *stages* that progressively admit negative levels.

Stage and Step Indices. We distinguish two levels of indexing in the redistribution process:

Within each stage M , the admissible tuples form a block that we denote by

$$(5.47) \quad \mathcal{B}_M \stackrel{\text{def}}{=} \{ \hat{s}^{(M,m)} : 0 \leq m \leq D - k_M^* \},$$

where k_M^* is the terminal position k^* after m many stages of mass redistribution and M is the number of total shifts (i.e. applications of the proposed shift function). Note also the added subscript on k^* . A detailed explanation will follow shortly; for the moment it suffices to observe that k^* shifts after each M , and this labeling records that dependence. By construction, $\hat{s}^{(M,0)} \equiv \hat{s}^{(M)}$ is the canonical seed for stage M , and successive elements are generated via

$$(5.48) \quad \hat{s}^{(M,m)} \rightsquigarrow \hat{s}^{(M,m+1)} \quad (0 \leq m < D - k_M^*).$$

as determined by our counting process. Thus each block $\mathcal{B}_M$ is a contiguous chain of tuples in lex order, beginning from the stage seed and ending when no further admissible redistribution is possible. In this way the full enumeration proceeds block-wise across stages M , while within each block it walks sequentially through the tuples that are generated over m many mass redistributions.

At stage $M = 1$ we “turn on” visits to -1 by minimally *migrating mass* from the rightmost admissible positive entries (i.e., from k_+ leftward down to 0) into the new coordinates that must be met first under the order, namely c_{-1} (and c_0 if needed to satisfy Equation (5.38)). This extension should be obvious to the reader as it is informed by the graphical structure just now adding in the negative side (c.f. Theorem 5.4). This produces a new seed tuple $\hat{s}^{(1)}$ that is lexicographically maximal among all tuples that reach -1 (no tuple greater than $\hat{s}^{(1)}$ in $\prec_{\text{lex } \pm}$ remains to be enumerated within this block, aside from those already handled at stage $M = 0$ which was the all positive case). Below we provide a sample case of this first mass shift that occurs for a general initial all positive seed tuple $\hat{s} \in \mathcal{C}_{D,k^*}^+$ and performs one mass shift. Note that these updates obey Equation (5.38)–Equation (5.40): at each step we remove two units of mass from the rightmost slot (or, if that slot contains only a single unit, from the next available slot as well) and redistribute this mass into the c_0 and c_{-1} positions. This procedure initializes the algorithm at the extremal path where -1 is the only negative value, though as the mass process continues, additional visits to -1 naturally appear. At the same time, the index k^* shifts one position to the right within the active block, while the slots with $c_k = 0$ are shifted one step to the left, a behavior we will make precise shortly.

$$(5.49) \quad \begin{aligned} \hat{s}^{(M=0, m=0)} &= \underbrace{(0, 0, \dots, 0, 0)}_{c_k, k \in [k_-, -1]}, \underbrace{1, 1, \dots, 1}_{c_0, k \in [0, k^* - 1]}, \underbrace{2, \dots, 2}_{c_k, k \in [k^*, k_+]} , (2 \text{ or } 1) \\ \hat{s}^{(M=1, m=0)} &= \underbrace{(0, 0, \dots, 0, 1)}_{c_k, k \in [k_-, -2]}, \underbrace{2, 1, \dots, 1}_{c_0, k \in [-1, k^* - 1]}, \underbrace{2, \dots, (2 \text{ or } 1)}_{c_k, k \in [k^*, k_+ - 1]}, \underbrace{0}_{k_+} \end{aligned}$$

We then enumerate all admissible tuples at stage $M = 1$ by the same mass redistribution walk as in the positive case, only now constrained by the newly included negative coordinate (and equivalently the new set of indices it is defined over). After exhausting this block, we repeat the same idea for $M = 2$: minimally migrate mass (again from the rightmost admissible positive entries) to satisfy the next required negative coordinate c_{-2} (together with the previously activated c_{-1} and the c_0 constraint), thereby producing a new seed $\hat{s}^{(2)}$, which is maximal for the -2 -admitting block under $\prec_{\text{lex } \pm}$. Continuing in this manner for $M = 3, 4, \dots, -k_-$, we *gradually* include $-3, -4, \dots, k_-$, each time starting from a canonical seed and sweeping the corresponding block in lex order.

In summary, the reader should picture a staircase of seed tuples:

$$(5.50) \quad \begin{aligned} \hat{s}^{(M=0, m=0)} &\rightsquigarrow \dots \rightsquigarrow \hat{s}^{(M=0, m=D-k_0^*)} \rightsquigarrow \hat{s}^{(M=1, m=0)} \rightsquigarrow \dots \rightsquigarrow \hat{s}^{(M=1, m=D-k_1^*)} \\ &\rightsquigarrow \hat{s}^{(M=2, m=0)} \rightsquigarrow \dots \rightsquigarrow \hat{s}^{(M=2, m=D-k_2^*)} \rightsquigarrow \dots \rightsquigarrow \hat{s}^{(M=-k_-, m=D-k_{-k_-}^*)} \end{aligned}$$

where each transition is effected by a minimal, right-to-left shift of mass that (i) enforces the next negative-side constraints from Equation (5.38)–Equation (5.40), and (ii) ensures the new seed is the *largest* element (in $\prec_{\text{lex } \pm}$) of its stage. Now we can formalize our shift function and write in our final closed-form combinatorial expression.

Formalizing the Shift Function. At the outer stage $M \in \{0, 1, \dots, -k_-\}$ we act only on the active index set Equation (5.41), realized as a left-translate of $[0, k_+]$ by M :

$$(5.51) \quad I_M \stackrel{\text{def}}{=} ([0, k_+] - M) \cap \mathbb{Z} = \{k - M : k \in [0, k_+]\} \cap \mathbb{Z} = [-M, k_+ - M] \cap \mathbb{Z}.$$

Equivalently, the blocks satisfy the recursion

$$(5.52) \quad I_{M+1} = I_M - 1 = \{k - 1 : k \in I_M\}.$$

Thus $I_0 = [0, k_+]$ and $I_1 = [-1, k_+ - 1]$, etc., i.e., the active set is pushed one unit to the left at each stage.

Remark 5.10. Position of k^* In global indices k^* is fixed, but its *position within the active block* I_M shifts right by one each stage. Measuring position from the left endpoint of I_M ,

$$(5.53) \quad \text{pos}_{I_M}(k^*) \stackrel{\text{def}}{=} k^* - \min I_M = k^* - (-M) = k^* + M.$$

Equivalently, in the locally re-centered (translated) coordinates

$$\tau_M : \mathbb{Z} \rightarrow \mathbb{Z}, \quad \tau_M(k) \stackrel{\text{def}}{=} k - \min I_M = k + M,$$

we have $\tau_M(I_M) = [0, k_+]$ and $\tau_M(k^*) = k^* + M$, making explicit that k^* advances one slot to the right at each outer stage M .

Given Equation (5.51)–Equation (5.52) and the conditions stated in Equation (5.38)–Equation (5.40), we now define the full process that each application of the shift function applies to the previous $\hat{s}$ in the chain starting at $\hat{s}^{(0)}$. We start at stage $M = 0$ with our initial seed tuple $\hat{s}^{(0)}$ with its associated parity tag $\beta \in \{1, 2\}$ as defined in Equation (5.25). We then define our shifting process formally by the update $\mathcal{A}_{M,t}$ as follows:

- **Odd case $\beta = 1$:**

- Input $\hat{s}_{\text{odd}}^{(M=0, m=0)} = (c_0, \dots, c_{k_+} = 1)$.
- Subtract 1 from c_{k_+} (so $c_{k_+} \leftarrow c_{k_+} - 1$).
- Form $v \stackrel{\text{def}}{=} (1, c_0, \dots, c_{k_+-1})$ (left-translate with a leading 1).
- Let $j_t \stackrel{\text{def}}{=} \max\{j : v_j = 2\}$ (rightmost local slot holding 2).
- Update

$$v_{j_t} \leftarrow 1, \quad v_1 \leftarrow v_1 + 1.$$

- Set $\hat{s}_{\text{odd}}^{(M=1, m=0)} \stackrel{\text{def}}{=} v$.

- **Even case $\beta = 2$:**

- Input $\hat{s}_{\text{even}}^{(M=0, m=0)} = (c_0, \dots, c_{k_+} = 2)$.
- Subtract 2 from c_{k_+} (so $c_{k_+} \leftarrow c_{k_+} - 2$).
- Form $v \stackrel{\text{def}}{=} (1, c_0, \dots, c_{k_+-1})$.
- Update

$$v_1 \leftarrow v_1 + 1.$$

$$- \text{Set } \hat{s}_{\text{even}}^{(M=1, m=0)} \stackrel{\text{def}}{=} v.$$

We then translate the active window to $I_{M+1} = I_M - 1$ and set the next seed. This process (i) reseeds the counting routine each time with a new $\hat{s}^{(i)}$, $i \in [0, -k_-]$, and (ii) produces both (a) a *geometric* terminal index showing where k^* sits in global coordinates, and (b) an *effective* index reflecting the mass reservoir used in the next counting pass. We make this explicit as follows.

Dynamic indices. We define the *geometric* terminal index

$$(5.54) \quad k_{\text{geo}}^*(M) \stackrel{\text{def}}{=} k^* + M,$$

which governs the placement of k^* inside I_M , and the *effective* terminal index

$$(5.55) \quad \tilde{k}(M) \stackrel{\text{def}}{=} k^* + 2M,$$

which governs the mass horizon available at stage M . After applying the stage update we set

$$\hat{s}^{(M+1)} = \mathcal{S}_M(\hat{s}^{(M)}),$$

and the next counting pass is performed with the reseated terminal index values $k_{\text{geo}}^*(M+1)$ and $\tilde{k}(M+1)$.

Reseeded counting at each stage. After each shift we rerun the positive-side counting with the reseated terminal index. Using the same closed form as in [Equation \(5.33\)](#) but with $k^* \mapsto k_{\text{geo}}^*(M)$ for geometry and $\tilde{k}(M)$ for the horizon, the *stage- M contribution* is

$$(5.56) \quad \mathcal{C}_{D, k_{\text{geo}}^*(M)}^w(\hat{s}^{(M)}) = \sum_{i=0}^{D-\tilde{k}(M)} \left\{ \binom{i+\ell - \left\lceil \frac{i+1}{2} \right\rceil - 1}{\ell - \left\lceil \frac{i+1}{2} \right\rceil - 1} + (\beta - 1) \binom{i+\ell - \left\lceil \frac{i+1}{2} \right\rceil - 1}{\ell - \left\lceil \frac{i+1}{2} \right\rceil - 1} \right\}$$

Here $(\beta - 1)$ is unchanged across stages by construction of our shifting process, and the upper limit $D - \tilde{k}(M) = D - k^* - 2M$ shrinks by 2 at each stage, reflecting the natural depletion of the mass reservoir by the shift update.

Within each stage we have the deterministic trajectory

$$\hat{s}^{(M,0)} \equiv \hat{s}^{(M)} \rightsquigarrow \hat{s}^{(M,1)} \rightsquigarrow \dots \rightsquigarrow \hat{s}^{(M,m)} \in \mathbb{N}^{I_M}, \quad m = 0, 1, \dots, m_{\text{max}}(M),$$

generated by the parity-dependent map

$$\hat{s}^{(M,m+1)} = \mathcal{S}_{\beta}^{(M)}(\hat{s}^{(M,m)}), \quad m = 0, 1, \dots, m_{\text{max}}(M) - 1,$$

where the counting index m coincides with the summation index i in [Equation \(5.56\)](#). We define the stage horizon and reseeding map by

$$m_{\text{max}}(M) \stackrel{\text{def}}{=} D - \tilde{k}(M), \quad \mathcal{S}^{(M)}(\hat{s}^{(M)}) \stackrel{\text{def}}{=} \hat{s}^{(M, m_{\text{max}}(M))}, \quad \hat{s}^{(M+1)} \stackrel{\text{def}}{=} \mathcal{S}^{(M)}(\hat{s}^{(M)}),$$

so that $m_{\text{max}}(M)$ is exactly the upper limit of the summation in [Equation \(5.56\)](#) and the terminal state of stage M becomes the seed for stage $M+1$.

Total reseeded count (using the existing outer stopping time)... A shift is applicable at stage M iff the active window can move left without colliding with either boundary; equivalently,

$$k_+ - M > k^* \quad \text{and} \quad -M > k_-.$$

Thus the outer process stops at the first M for which a further shift would not change k^* , namely

$$(5.57) \quad T \stackrel{\text{def}}{=} \min\{k_+ - k^*, -k_-\}.$$

For $M < T$ we have $k_{\text{geo}}^*(M) = k^* + M$ and the next reseed is well-defined; at $M = T$ no further shift is applied and k^* ceases to change. By recursion, $\hat{s}^{(0)}$ is the given seed and, for $M \geq 1$, $\hat{s}^{(M)} = \mathcal{S}^{(M-1)}(\hat{s}^{(M-1)})$.

We then define the total reseeded count as

$$(5.58) \quad \mathcal{C}_{D,\text{dyn}}^w(\hat{s}^{(0)}) = \sum_{M=0}^T \mathcal{C}_{D,k_{\text{geo}}^*(M)}^w(\hat{s}^{(M)}), \quad k_{\text{geo}}^*(M) = k^* + M, \quad \tilde{k}(M) = k^* + 2M.$$

Remark 5.11 (Note on the naturality of our stopping condition). Our stopping index T is not imposed externally but arises from the natural mechanics of the shift process $\mathcal{A}_M$. The active window can shift left exactly T times before either the boundary k_- or the right-capacity k_+ prevents a further shift. With our effective index $\tilde{k}(M) = k^* + 2M$ the available mass horizon shrinks by 2 per completed shift and the final “leftover” unit is automatically counted when $m_{\text{max}}(M) = 1$ (even case $\cup \{k^* = 0\}$) or $m_{\text{max}}(M) = 0$ (odd case), so no additional ad hoc correction is required.

6. Computational Complexity Analysis. Before we begin our computational complexity analysis analytically, it serves to observe something that should appear intuitive. We start by referencing a visualization of a recombining tree from [16]:

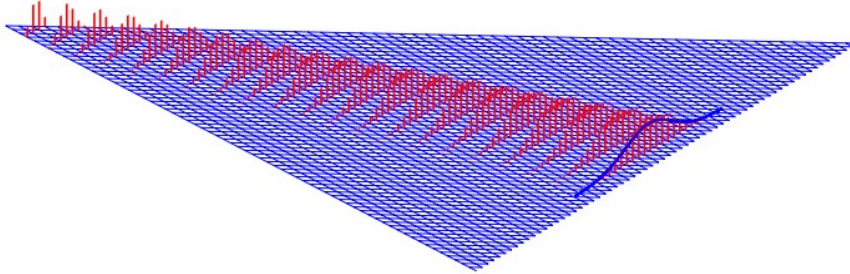


Fig. 7: Distribution of the Concentration of Paths

Although this visualization is of a binomial tree, it highlights a phenomenon that our counting algorithm establishes analytically and empirically: the path density at each terminal position k^* peaks at $k^* = 0$ and smoothly decays to 1 at the boundary terminal nodes $k^* = \{-D\}, \{D\}$. This observation is rigorously justified through the symmetry result in [Theorem 5.8](#) and is true for any recombining n -nomial tree (which of course includes the tree we worked over). Consider the distribution of terminal nodes indexed by k^* in the interval

$$[k^* = -D, 0) \cup \{0\} \cup (0, D].$$

From [Figure 7](#) (or equivalently by direct evaluation of [Equation \(5.58\)](#) for each k^*), it follows that the cardinalities satisfy

$$(6.1) \quad \max_{k^* \in [-D, D]} |\{\mathbf{p} \in \mathcal{C}_{D,k^*}\}| = |\{\mathbf{p} \in \mathcal{C}_{D,0}\}|,$$

with the path counts decaying symmetrically as $|k^*|$ increases. In particular, the boundary values obey

$$(6.2) \quad \forall D \in \mathbb{N}, \quad |\{\mathbf{p} \in \mathcal{C}_{D, \pm D}\}| = 1,$$

so that the distribution is unimodal at $k^* = 0$ and strictly decreasing toward the end-points. Thus, in analyzing the computational complexity of the algorithm that performs computations over these paths, it suffices to show that the runtime is bounded above by the enumeration count of the path family with the largest cardinality. Equivalently, this reduces to identifying the terminal index k^* at which the maximum mass accumulates—both perspectives are equivalent. Accordingly, we bound the counting formula by considering the set $\mathcal{C}_{D,0}$, since

$$(6.3) \quad \max_{k^* \in [-D, D]} |\mathcal{C}_{D, k^*}| = |\mathcal{C}_{D, 0}|.$$

On a per-round basis, the largest enumeration occurs when we generate all positive paths i.e. $\mathcal{C}_{D,0}^+$. (Note that the symmetric contribution from negative paths is picked up in parallel, but is always strictly smaller because the mass is constrained at each application of the shift function.) To formalize this, we let

- $\ell \stackrel{\text{def}}{=} \lceil D/2 \rceil$ - "slot" parameter used in every binomial index
- $s \stackrel{\text{def}}{=} \lfloor D/2 \rfloor$ - the maximum number of "shifts" the parity-aware shift operator $\mathcal{S}_\beta$ can perform

We can that that each general configuration of an arbitrary $\hat{c} = (c_0, \dots, c_{k^*}) \in \mathcal{C}_{D, k^*}^+$ is counted by Equation (5.33). We can thus make the substitutions

- $r_i \stackrel{\text{def}}{=} \ell - \lceil \frac{i+1}{2} \rceil$
- $N_i \stackrel{\text{def}}{=} i + r_i - 1$
- $K_i \stackrel{\text{def}}{=} r_i - 1$

into Equation (5.33) to get:

$$\mathcal{C}(s^{(0)}) = \sum_{i=0}^{D-k^*} \left(\binom{N_i}{K_i} + (\beta - 1) \binom{N_i}{K_i + 1} \right).$$

One application of the shift operator $\mathcal{S}_\beta$ removes two units of mass from the right-most end and slides the tuple one place right. Therefore, after the k^{th} shift, we conservatively have:

$$D_k \stackrel{\text{def}}{=} D - k$$

We can then present the following lemma:

LEMMA 6.1 (Refined upper bound via entropy). *Let c be a configuration with effective depth $\tilde{d} \in \mathbb{N}$ and put $\ell = \lceil \tilde{d}/2 \rceil$. For $i \geq 0$ define*

$$(6.4) \quad \begin{aligned} r_i &= \ell - \left\lceil \frac{i+1}{2} \right\rceil, & N_i &= i + r_i - 1 = \ell - 1 + i - \left\lceil \frac{i+1}{2} \right\rceil, \\ K_i &= r_i - 1 = \ell - \left\lceil \frac{i+1}{2} \right\rceil - 1. \end{aligned}$$

Then there exists a constant $C > 0$ such that

$$(6.5) \quad \mathcal{C}(\hat{s}^{(0)}) \leq C \tilde{d}^{1/2} \gamma^{\tilde{d}}, \quad \gamma = 2^{\frac{3}{4}H(1/3)} \approx 1.61185,$$

where $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$.

Proof. Use $\lceil x \rceil \geq x$ and $\lceil x \rceil \leq x + 1$ to get, for all relevant i ,

$$N_i \leq \ell - 1 + \frac{i}{2} \leq \frac{3}{4}\tilde{d} + \mathcal{O}(1), \quad \frac{K_i}{N_i} \geq \frac{2\ell - i - 3}{2\ell + i - 2} = \frac{1}{3} - \mathcal{O}\left(\frac{1}{\tilde{d}}\right).$$

By the entropy-form binomial bound (e.g., [10, Thm. VIII.1], [5, Ch. 11]),

$$\binom{N}{K} \leq \frac{C_0}{\sqrt{N}} 2^{NH(K/N)}.$$

Apply with $(N, K) = (N_i, K_i)$ and the bounds above to obtain

$$\binom{N_i}{K_i}, \binom{N_i}{K_i + 1} \leq \frac{C_1}{\sqrt{\tilde{d}}} \gamma^{\tilde{d}}, \quad \gamma = 2^{\frac{3}{4}H(1/3)}.$$

Summing over at most $\tilde{d} + 1$ indices i yields $\mathcal{C}(\hat{s}^{(0)}) \leq C \tilde{d}^{1/2} \gamma^{\tilde{d}}$. $\square$

Remark 6.2 (Optional collapse). Using the hockey-stick identity $\sum_{i=0}^r \binom{i+\ell-1}{\ell-1} = \binom{r+\ell}{\ell}$ [11, Eq. (5.25)] to collapse the i -sum first gives a single binomial term and improves the pre-factor to $\mathcal{O}(\tilde{d}^{-1/2})$ (same base γ).

We then write another Lemma where we examine the per-round decay under shifts:

LEMMA 6.3 (Per-round decay under shifts). *Let $\mathcal{C}_k$ denote the cost after exactly k applications of $\mathcal{S}_\beta$, and set $D_k \stackrel{\text{def}}{=} D - k$. There exists a constant $C > 0$ (the same as in Lemma 6.1) such that, for all $0 \leq k \leq m$,*

$$(6.6) \quad \mathcal{C}_k \leq C D_k^{1/2} \gamma^{D_k} \leq C D^{1/2} \gamma^D \rho^k, \quad \rho \stackrel{\text{def}}{=} \gamma^{-1} \in (0, 1).$$

Proof. After k shifts the effective depth is at most $D_k = D - k$. Applying Lemma 6.1 with $\tilde{d} = D_k$ yields $\mathcal{C}_k \leq C D_k^{1/2} \gamma^{D_k}$. Since $D_k^{1/2} \leq D^{1/2}$ and $\gamma^{D_k} = \gamma^D \cdot \gamma^{-k}$, we obtain Equation (6.6) with $\rho = \gamma^{-1}$. $\square$

We then have yet another Lemma where we examine the per-round decay under shifts:

LEMMA 6.4 (Outer stopping time). *The shift process stops no later than round $s = \lfloor D/2 \rfloor$.*

Proof. By definition, one application of $\mathcal{S}_\beta$ via $\mathcal{A}$ removes two units of mass from the rightmost end and shifts the tuple one slot to the right (c.f. Subsection 5.1.1). Let M_0 denote the total removable mass available to the shift operator at the start of a global round. Because the tuple encodes a path of length d with at least one unit in every occupied slot, the removable mass satisfies $M_0 \leq D$. Each shift reduces the removable mass by exactly 2 and never increases it (collisions merge “2”s but do not create new ones). Hence after at most $\lfloor D/2 \rfloor$ shifts the removable mass is exhausted and no further application of $\mathcal{S}_\beta$ is possible. Equivalently, the pivot cannot advance beyond the rightmost slot once M_0 has been depleted, so the procedure halts by round $s = \lfloor D/2 \rfloor$. $\square$

We then clearly have a complete theorem for the computational upper bound on the combinatorics:

THEOREM 6.5 (Total running time). *Let*

$$T(D) \stackrel{\text{def}}{=} \sum_{k=0}^s \mathcal{C}_k, \quad s \stackrel{\text{def}}{=} \lfloor D/2 \rfloor,$$

where $\mathcal{C}_k$ is the cost after exactly k shifts. With $\gamma \stackrel{\text{def}}{=} 2^{\frac{3}{4}H(1/3)} \approx 1.61185$ and $\rho \stackrel{\text{def}}{=} \gamma^{-1} \approx 0.6206$, [Lemma 6.1](#) and the per-round decay lemma give

$$T(D) \leq C D^{1/2} \gamma^D \sum_{k=0}^s \rho^k \leq \frac{C}{1-\rho} D^{1/2} \gamma^D.$$

Numerically, $1 - \rho \approx 0.3794$ and $\frac{1}{1-\rho} \approx 2.636$, hence

$$T(D) \leq C_* D^{1/2} \gamma^D \quad \text{with} \quad C_* \stackrel{\text{def}}{=} \frac{C}{1-\rho} \lesssim 2.64 C.$$

In particular,

$$T(D) = \mathcal{O}(D^{1/2} 1.612^D).$$

Proof. By the per-round bound, $\mathcal{C}_k \leq C D^{1/2} \gamma^D \rho^k$ for $0 \leq k \leq s$. Summing the geometric series and using $s = \lfloor D/2 \rfloor$ (so the tail factor drops),

$$T(D) \leq C D^{1/2} \gamma^D \frac{1 - \rho^{s+1}}{1 - \rho} \leq \frac{C}{1 - \rho} D^{1/2} \gamma^D.$$

Insert the numerics for $\rho = \gamma^{-1}$. □

COROLLARY 6.6 (Exponential speedup over naive recursion). *The naive exhaustive recursion costs 3^D operations. Therefore*

$$\frac{3^D}{T(D)} \geq \frac{1}{C_*} \frac{(3/\gamma)^D}{D^{1/2}} \quad \text{with} \quad \frac{3}{\gamma} \approx 1.861.$$

Hence the ratio grows unboundedly like $(1.861)^D / (C_* D^{1/2})$, proving an exponential improvement.

These bounds establish a rigorous computational upper bound on the complexity of the combinatorial enumeration. In turn, they also imply a theoretical lower bound on the runtime that any implementation of our algorithm must incur. The key task then is to analyze the performance of our actual implementation, derive its achievable practical upper bound, and compare this with the theoretical lower bound. The gap between the two quantifies how closely the implemented algorithm approaches the fundamental efficiency limits.

7. Conclusions. We close by highlighting two complementary threads. In our first thread *future work*, we outline several directions that naturally follow from our framework, including extensions to general n -nomial trees, links to Motzkin or Dyck-style occupation profiles, sharper complexity bounds, and density-aware sampling schemes for very deep trees. Together, these closing sections synthesize the practical impact of our contributions and chart a concrete agenda for subsequent research. Then in our second thread *applications*, we summarize how our enumeration

and counting results can be used in practice—e.g., discretizing stochastic dynamics on lattices, path-dependent valuation and risk aggregation, and planning/rollout in various discrete event problems—emphasizing when the constructive algorithm is preferable to classical recursion.

7.1. Further Work.

7.1.1. Connection with Motzkin Paths. There is a natural correspondence between length- D walks on our recombining trinomial tree (step set $\{-1, 0, +1\}$) and Motzkin walks of length D (directed lattice paths with up, flat, down steps). Imposing the usual half-plane constraint (never going below the baseline) and endpoint 0 specializes these to classical Motzkin paths, whose univariate generating function is algebraic and whose enumerants are the Motzkin numbers. Removing 0-steps further specializes to Dyck paths (Catalan objects). See, e.g., Banderier–Flajolet for a unified treatment of directed lattice paths (including $\{-1, 0, +1\}$) via the kernel method, generating functions, and half-plane constraints [2].

What is distinctive in our setting is the *cardinality tuple* (occupation profile) $c(\pi) = (c_k)_{k \in [k_-, k_+]}$, which records the number of visits to each level along a path. This refines beyond standard Motzkin/Dyck statistics (peaks, returns, level steps at height h , etc.) typically used for Narayana- or Fine-type refinements. A promising direction is to relate our multilevel occupation profiles to *Motzkin polynomials* (a multivariate scheme that weights steps by height) and to continued-fraction/J-fraction representations: Oste–Van der Jeugt develop Motzkin polynomials and show how tridiagonal-matrix powers and weighted Motzkin paths are encoded by such generating functions [19].

Open questions w.r.t the Connection with Motzkin Paths.

- (Q1) Does the multivariate series $F(\mathbf{y}; t) = \sum_{D \geq 0} \sum_{\pi} t^D \prod_k y_k^{c_k(\pi)}$ admit a closed J-fraction/continued-fraction form that matches a suitable specialization of Motzkin polynomials? If so, our weak-composition formula for fixed $c(\pi)$ could potentially follow by coefficient extraction.
- (Q2) Under nonnegativity constraints (meanders/excursions), can one potentially obtain Narayana-type refinements that *condition* on $c(\pi)$ (e.g., total visits at nonnegative levels) and compare them to known refinements for Motzkin and or Dyck paths?
- (Q3) How do endpoint constraints ($k^* \neq 0$) and parity rules interact with the standard first-return/first-step decompositions used for Motzkin paths? Can these be expressed as simple functional equations for $F(\mathbf{y}; t)$?

We emphasize that, while the Motzkin correspondence is classical, we are not aware of a prior closed enumeration by the *full* level-visit profile $c(\pi)$ on trinomial trees. Establishing whether our occupation-profile enumeration reduces to (or strictly extends) known Motzkin polynomial frameworks is an interesting avenue for future work.

7.1.2. Potential for Gray-Code Optimization. Our algorithm possesses a strong potential for further optimization. Here, we make a loose remark with some loose observations on bounds to theorize on how one could actually go about implementing gray codes to optimize the algorithm given its construction:

Remark 7.1 (Further optimization via Gray codes and constant-delay generation). Our recursionless design updates only a constant number of tuple entries per emitted object (an adjacent unit transfer), so the inner enumeration can be imple-

mented with a *minimal-change* (Gray) successor that guarantees *constant worst-case delay* and $O(1)$ extra workspace beyond the current state [8, 21, 23]. Let $c_{\text{succ}} > 0$ denote this per-output constant.

Cost decomposition. Write the implementation cost as

$$T_{\text{impl}}(D) = T_{\text{outer}}(D) + T_{\text{inner}}(D).$$

By the stopping-time bound, the outer routine applies $\mathcal{S}_\beta$ at most $s = \lfloor D/2 \rfloor$ times. If we conservatively initialize the negative-path seed in $\Theta(D)$ time per stage (Algorithm C.1), then

$$T_{\text{outer}}(D) \leq c_{\text{init}} D s + c_{\text{shift}} s = \mathcal{O}(D^2) \quad (\text{prototype bound}).$$

With the standard pointerized update (no full re-scan), the same work is $O(1)$ per stage, hence $T_{\text{outer}}(D) = O(D)$ (achievable).

For the inner enumeration, a Gray successor on the constrained weak compositions yields

$$T_{\text{inner}}(D) \leq c_{\text{succ}} \left| \mathcal{C}_{D, \text{dyn}}^w(\hat{s}^{(0)}) \right| = c_{\text{succ}} \cdot \Theta(D^{1/2} \gamma^D),$$

by Theorem 6.5. Therefore

$$T_{\text{impl}}(D) \leq \underbrace{c_{\text{succ}} \Theta(D^{1/2} \gamma^D)}_{\text{output-sensitive, dominant}} + \underbrace{\mathcal{O}(D^2)}_{\text{prototype init + shifts}} = \Theta(D^{1/2} \gamma^D),$$

and with $O(1)$ -time reseeding the additive term improves to $O(D)$, which is negligible compared to $D^{1/2} \gamma^D$.

Takeaway. Even without changing the mathematics, a loopless Gray-code successor makes the implementation *output-optimal*: constant worst-case delay per emitted tuple and total time within a constant factor of the theoretical lower bound implied by the combinatorial count. Practically, the inner loops can be replaced by a streaming `next()` that touches only two adjacent entries per step, while the outer loop performs at most $s = \lfloor D/2 \rfloor$ constant-time reseeds.

7.1.3. Extension to n -nomial Recombining Trees. We also believe there is a closed-form enumeration for general n -nomial recombining trees—depending on depth D , terminal index k^* , and branch multiplicity n —with a corresponding complexity bound of the form $\mathcal{O}(D b(n)^D)$. Deriving it appears to require a fully multivariate occupation-profile framework and new symmetry reductions beyond the trinomial case as well as a more nuanced mass-redistribution process; pursuing these technicalities is beyond the scope of this paper and is left as future work.

7.2. Applications. Our results are directly useful in several settings:

(A) *Option pricing on recombining trees.* The constructive enumeration supports exact valuation of path-dependent claims (e.g., Asian, barrier, lookback) by aggregating payoffs over occupation profiles (cardinality tuples). Closed-form counts enable stratified/importance sampling and reduce variance; early exercise (American-style) fits via dynamic programming on the enumerated successor sets. Listing these also allows for analysis of systemic risk when run in conjunction with known algorithms like FSG (forward-shooting grid).

(B) *Discrete-event control and scheduling.* Event sequences in queues or inventory systems can be modeled as trinomial walks; the cardinality tuple records level visits (e.g., buffer or stock levels). This yields exact reachability distributions, cost aggregation along trajectories, and worst-case or risk-sensitive evaluations under resource constraints.

(C) *Planning and model-based reinforcement learning.* Finite-horizon rollouts on the recombining tree avoid recursion and duplicates, while occupation profiles serve as compact trajectory features for value expansion or policy improvement. For long horizons, the same structure supports density-aware pruning or stratified sampling with explicit error control.

In moderate-depth regimes requiring auditability and tight error budgets, the proposed enumeration is preferable to naive recursion or unconstrained Monte Carlo; for very deep horizons, it remains a principled backbone for hybrid sampling schemes.

Appendix A. Generation of Non-Unique Combinations.

Algorithm A.1 Generate Combinations of Paths using a Recursive DFS Procedure

```

1: function GENERATECOMBINATIONSVIARECURSIVEDFS( $d, k$ ) {/* DFS recursion to generate all paths of
   depth  $d$  that terminates at  $k$  */}
2:   All Paths  $\leftarrow$  Empty List
3:   Current Path  $\leftarrow$  Empty List {Start DFS from the root node}
4:   DFS(0, 0, Current Path, All Paths,  $d, k$ )
5: end function
6: Return: All Paths
7: function DFS(current-depth, current-position, current-path, All-Paths,  $d, k$ ) {/* DFS Recursion;
   current-position  $\in [-\text{current-depth}, +\text{current-depth}]$  */}
8:   if current-depth ==  $d$  then
9:     Add current-path to All-Paths if it terminates on  $k$ 
10:    Return
11:   end if
12:   Append “(-1,current-depth)” to current-path {/* Recursion-step for -1 */}
13:   DFS(current-depth+1, current-position+1, current-path, All-Paths,  $d, k$ )
14:   Remove last element in current-path
15:   Append “(0,current-depth)” to current-path {/* Recursion-step for 0 */}
16:   DFS(current-depth+1, current-position+1, current-path, All-Paths,  $d, k$ )
17:   Remove last element in current-path
18:   Append “(+1,current-depth)” to current-path {/* Recursion-step for +1 */}
19:   DFS(current-depth+1, current-position+1, current-path, All-Paths,  $d, k$ )
20:   Remove last element in current-path
21: end function

```

Algorithm A.2 Generate Combinations of Paths using Hashing

```

1: function GENERATECOMBINATIONS(input_map, length, buf, cbuf)
2:   if length == 1 then
3:     path  $\leftarrow$  EXTRACTPATHFROMBUFFER(buf)
4:     STOREPATH(path)
5:     return
6:   end if
7:   if cbuf == buf then
8:     cbuf[0]  $\leftarrow$  {key: 0, value: input_map[0]}
9:     GENERATECOMBINATIONS(input_map, length - 1, buf, cbuf + 1)
10:    return
11:   end if
12:   prev_key  $\leftarrow$  cbuf[-1].key
13:   for offset in [-1, 0, 1] {Traverse Child Nodes} do
14:     next_key  $\leftarrow$  prev_key + offset
15:     if next_key in input_map then
16:       cbuf[0]  $\leftarrow$  [key: next_key, value: input_map[next_key]]
17:       GENERATECOMBINATIONS(input_map, length - 1, buf, cbuf + 1)
18:     end if
19:   end for
20: end function
21: Return: output_storage containing all generated paths

```

Algorithm A.3 Recursion-free Generation of Paths

```

1: function RECURSIONFREEGENERATECOMBINATIONS( $D, k^*$ ) {/* Recursion-free procedure to generate all
   paths of depth  $D$  that terminate at depth  $D$  at  $(k^*, D)^*$ */}
2:   current path  $\leftarrow$  The path that contains the maximally reachable node  $(p_{k^*})'$  Equation (4.2) for the
   positive-paths, using Init-Array(depth, terminal_node, path) in Appendix B.1 {/* This path starts
   from the root-node and terminates at depth  $D$ , contains  $p_{k^*}$ ', and has no negative elements */}
3:   All Paths = {current path}
4:   ((Boolean) tick-down, tick-down-index) = ComputeTickDown(current path) {/* Returns False,
   along with the largest index that can be ticked-down; returns False if none found */}
5:   while tick-down do
6:     Replace  $(m, \text{tick-down-index})$  with  $(m-1, \text{tick-down-index})$  in current path {/* Decrement the
     tick-down-index by 1 */}
7:     All Paths = All Paths  $\cup$  current-path {/* Add the ticked-down, valid, path to the set of paths
     */}
8:     for  $i \in \{\text{tick-down-index}+1, d\}$  do
9:       if CheckValidity(current path,  $i, +1$ ) then
10:        Replace  $(\text{value}, i) \in$  current path with  $(\text{value}+1, i)$ . {/* Explore indices larger than
        tick-down-index to find other valid paths */}
11:        All Paths = All Paths  $\cup$  current-path.
12:      end if
13:    end for
14:    ((Boolean) tick-down, tick-down-index) = ComputeTickDown(current path)
15:  end while
16: end function
17: Return: All Paths
18: function CHECKVALIDITY(current path, index, change) {/* change  $\in \{-1, 1\}$ , increment or decrement
   */}
19: Replace  $(\text{value}, \text{index}) \in$  current path with  $(\text{value}+\text{change}, \text{index})$  and check if the new path satisfies
   the vectorized rules in Section 2. Return True if it does, else Return False
20: end function
21: function COMPUTETICKDOWN(current path)
22: tick-down-index is the largest value in  $\{0, 1, \dots, d\}$  such that  $(m, \text{tick-down-index}) \in$  current path
   and CheckValidity(current path,  $m, -1$ ) = True
23: Return: (True, tick-down-index) if found; else Return: (False, NaN)
24: end function

```

Appendix B. Path Init Algorithm.**Algorithm B.1** Initialize Path Array

```

1: function INIT-ARRAY( $D, k^*$ )
2: for  $i = 1$  to depth - 1 do
3:   if  $i < \text{max\_node\_pos} + 1$  then
4:     path[ $i-1$ ]  $\leftarrow i$ 
5:   else if depth mod 2 == terminal_node mod 2 then
6:     for  $j = \text{max\_node\_pos} - 1$  down to terminal_node + 1 do
7:       path[ $i-1$ ]  $\leftarrow j$ 
8:        $i \leftarrow i + 1$ 
9:     end for
10:     $i \leftarrow i - 1$  {Adjust for loop increment}
11:   else
12:     path[ $i-1$ ]  $\leftarrow \text{max\_node\_pos}$ 
13:      $i \leftarrow i + 1$ 
14:     for  $j = \text{max\_node\_pos} - 1$  down to terminal_node + 1 do
15:       path[ $i-1$ ]  $\leftarrow j$ 
16:        $i \leftarrow i + 1$ 
17:     end for
18:      $i \leftarrow i - 1$  {Adjust for loop increment}
19:   end if
20: end for
21: end function

```

Appendix C. Negative Path Init.

Algorithm C.1 Shift-and-Reseed ($\mathcal{A}_{M,t}$) — one stage

```

1: {/*  $k_l$  = leftmost index of active window;  $k_r$  = rightmost index of active window. Ev-
   everything shifts one step to the left each iteration. */}
2: Input:  $D, \hat{s}^{(i)} = (c_{k_l}, \dots, c_{k_r}), k_l, k_r, k^*$ 
3: Output:  $\hat{s}^{(i+1)}, k_l - 1, k_r - 1, k^* + 1$ 
4: /* Apply (5.1.1) on  $I_M = [k_l, k_r]$ . Shift  $k^*$  to the right by one index */
5:  $\beta \leftarrow c[k_r]$  /* usually 0, 1, 2; at  $k^* = 0$  may be 3. */
6: if  $\beta = 0$  then
7:   return  $(\hat{s}^{(i)}, k_l - 1, k_r - 1, k^* + 1)$ 
8: end if
9: if  $(\beta = 3) \wedge (k^* = 0)$  then
10:   $\beta \leftarrow 2$ 
11: end if
12:  $c[k_r] \leftarrow c[k_r] - \beta$  /* consume at right edge */
13: LET  $L \leftarrow k_r - k_l + 1$ ; BUILD  $v[0..L - 1]$ ;  $v[0] \leftarrow 1$ 
14: for  $j \leftarrow 1$  to  $L - 1$  do
15:    $v[j] \leftarrow c[k_l + j - 1]$ 
16: end for
17: if  $\beta = 1$  then
18:   for  $j \leftarrow L - 1$  downto 1 do
19:     if  $v[j] = 2$  then
20:        $v[j] \leftarrow 1$ ;
21:       break
22:     end if
23:   end for
24:    $v[1] \leftarrow v[1] + 1$ 
25: else if  $\beta = 2$  then
26:    $v[1] \leftarrow v[1] + 1$ 
27: end if
28: for  $j \leftarrow 0$  to  $L - 1$  do
29:    $\hat{s}^{(i+1)}[k_l - 1 + j] \leftarrow v[j]$ 
30: end for
31: return  $(\hat{s}^{(i+1)}, k_l - 1, k_r - 1, k^* + 1)$ 

```

Appendix D. Generate All Unique Paths.

Algorithm D.1 Unique Path Generation (Reseeded, recursion-free, with starting-tuple invariants)

```

1: function GEN-UNIQUE-COMB( $D, \hat{s}^{(0)}, k_l, k_r, k^*$ ) {/*  $k_l$  = leftmost index of active window;  $k_r$  =
   rightmost index. Each stage shifts the window one step left. */}
2: Require:
   •  $\hat{s}^{(0)}$  built via Init-Array(depth, terminal_node, path) in Algorithm B.1.
   • Stage shift  $\mathcal{S}_M$  and parity map  $\mathcal{S}_\beta^{(M)}$  in Algorithm C.1.

   {/* Starting-tuple structure and invariants (cf. Equation (5.9)):
   
$$\hat{s}^{(0)} = \left( \underbrace{\omega_1}_{0 \leq j < k^*} \circ \underbrace{\omega_2 \circ \omega_3 \circ \omega_4}_{\substack{j=k^* \quad k^* < j < k_r \quad j=k_r}} \right).$$

   */}

3:  $T \leftarrow \min\{k_r - k^*, -k_l\}$ 
4: total  $\leftarrow 0$ 
5: for  $M \leftarrow 0$  to  $T$  do
6:   {/* Dynamic terminal indices */}
7:    $k_{\text{geo}}^* \leftarrow k^* + M$ 
8:   {/* placement of  $k^*$  inside current window  $I_M = [k_l, k_r]$  */}
9:    $\tilde{k} \leftarrow k^* + 2M$ 
10:  {/* mass horizon available at stage  $M$  */}
11:   $m_{\text{max}} \leftarrow D - \tilde{k}$ 
12:  {/* Stage- $M$  counting pass with reseeded terminal index (cf. (5.56)) */}
13:  stage_contrib  $\leftarrow 0$ 
14:   $\hat{s}^{(M,0)} \leftarrow \hat{s}^{(M)}$ 
15:  for  $m \leftarrow 0$  to  $m_{\text{max}}$  do
16:    {/* closed form with  $k^* \mapsto k_{\text{geo}}^*$  and horizon  $\tilde{k}$  */}
17:    stage_contrib  $+= \binom{m + \ell - \lceil \frac{m+1}{2} \rceil - 1}{\ell - \lceil \frac{m+1}{2} \rceil - 1} + (\beta - 1) \binom{m + \ell - \lceil \frac{m+1}{2} \rceil - 1}{\ell - \lceil \frac{m+1}{2} \rceil}$ 
18:    if  $m < m_{\text{max}}$  then
19:       $\hat{s}^{(M,m+1)} \leftarrow \mathcal{S}_\beta^{(M)}(\hat{s}^{(M,m)})$ 
20:      {/* deterministic in-stage update; Algorithm C.1 */}
21:      {/* Respect invariants: slots in the translated fixed block (preimage of  $\omega_1$ ) remain untouched;
         redistribution uses the translated  $\omega_2 \circ \omega_3 \circ \omega_4$ . */}
22:    end if
23:  end for
24:  total  $+=$  stage_contrib
25:  {/* Reseed next stage from terminal in-stage state; then shift the window left */}
26:  if  $M < T$  then
27:     $\hat{s}^{(M+1)} \leftarrow \hat{s}^{(M,m_{\text{max}})}$ 
28:    {/* i.e.  $\mathcal{S}^{(M)}(\hat{s}^{(M)})$  */}
29:     $(k_l, k_r) \leftarrow (k_l - 1, k_r - 1)$ 
30:    {/*  $I_{M+1} = I_M - 1$ ; the fixed/redistributor partition reappears translated */}
31:    {/* Geometric index advances next loop by  $M \mapsto M + 1$ ; base  $k^*$  remains unchanged. */}
32:  end if
33: end for
34: return total {/* equals  $\sum_{M=0}^T \mathcal{C}_{D,k_{\text{geo}}^*(M)}^w(\hat{s}^{(M)})$  with  $k_{\text{geo}}^*(M) = k^* + M, \tilde{k}(M) = k^* + 2M$ . */}
35: end function

```

Acknowledgments. We would like to acknowledge the assistance of Manav Vora, Ryan Roach, and Pingbang Hu of the University of Illinois, Urbana-Champaign who gave me some useful notes on writing my manuscript and notes on various proofs. I would also like to thank William Cosley from the University of Northern Iowa who helped me write Appendix A that helped me first visualize the path enumeration of

the traditional approach. Finally, I'd like to acknowledge Dāniels Ponamarjovs from the College of Alberta, Latvia who helped me throughout the initial process when it came to proper C++ coding practices.

REFERENCES

- [1] J. AHN AND M. SONG, *Convergence of the trinomial tree method for pricing european/american options*, Applied Mathematics and Computation, 189 (2007), pp. 575–582, <https://doi.org/10.1016/j.amc.2006.11.132>.
- [2] C. BANDERIER AND P. FLAJOLET, *Basic analytic combinatorics of directed lattice paths*, Theoretical Computer Science, 281 (2002), pp. 37–80, [https://doi.org/10.1016/S0304-3975\(02\)00007-5](https://doi.org/10.1016/S0304-3975(02)00007-5), https://lipn.univ-paris13.fr/~banderier/Papers/tcs_banderier_flajolet_2002.pdf.
- [3] Y. CHENG, *Generating combinations by three basic operations*, Journal of Computer Science and Technology, 22 (2007), pp. 909–913, <https://doi.org/10.1007/s11390-007-9094-7>.
- [4] P. CLIFFORD AND O. ZABORONSKI, *Pricing options using trinomial trees*, tech. report, University of Warwick, 2008, <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=6b973e5ecae75a61bf4add2efe33c4c6356210de>. Accessed April 15, 2025.
- [5] T. M. COVER AND J. A. THOMAS, *Elements of Information Theory*, Wiley, 2 ed., 2006.
- [6] T. F. CRACK, *Trinomial option pricing: A primer*, tech. report, University of Otago, Department of Accountancy and Finance, Nov. 2024, <https://doi.org/10.2139/ssrn.4955216>, <https://ssrn.com/abstract=4955216>. SSRN Working Paper No. 4955216. Accessed April 15, 2025.
- [7] T.-S. DAI, *Pricing asian options on lattices*, master's thesis, National Taiwan University, Taipei, Taiwan, 2000, https://www.csie.ntu.edu.tw/~lyuu/theses/thesis_d88006.pdf. Accessed April 15, 2025.
- [8] P. EADES AND B. D. MCKAY, *An algorithm for generating subsets of fixed size with a strong minimal change property*, Information Processing Letters, 19 (1984), pp. 131–133.
- [9] G. EHRLICH, *Loopless algorithms for generating permutations, combinations, and other combinatorial configurations*, Journal of the ACM, 20 (1973), pp. 500–513, <https://doi.org/10.1145/321765.321781>.
- [10] P. FLAJOLET AND R. SEDGEWICK, *Analytic Combinatorics*, Cambridge University Press, 2009.
- [11] R. L. GRAHAM, D. E. KNUTH, AND O. PATASHNIK, *Concrete Mathematics*, Addison–Wesley, 2 ed., 1994.
- [12] E. T. IRONS AND W. FEURZEIG, *Comments on the implementation of recursive procedures and blocks in ALGOL 60*, Communications of the ACM, 4 (1961), pp. 65–69, <https://doi.org/10.1145/366062.366090>.
- [13] S. M. JOHNSON, *Generation of permutations by adjacent transposition*, Mathematics of Computation, 17 (1963), pp. 282–285, <https://doi.org/10.1090/S0025-5718-1963-0159764-2>.
- [14] D. E. KNUTH, *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*, Addison-Wesley Professional, 2011.
- [15] D. E. KNUTH, *The Art of Computer Programming, Volume 4B: Combinatorial Algorithms*, Addison-Wesley Professional, 2022.
- [16] Y. LEBEDEV AND A. BANERJEE, *Gaussian recombining split tree*. arXiv preprint arXiv:2405.16333, 2024, <https://doi.org/10.48550/arXiv.2405.16333>, <https://arxiv.org/abs/2405.16333>. q-fin.CP.
- [17] J. MA AND T. ZHU, *Convergence rates of trinomial tree methods for option pricing under regime-switching models*, Applied Mathematics Letters, 39 (2015), pp. 13–18, <https://doi.org/10.1016/j.aml.2014.07.020>.
- [18] D. MICHIE, “Memo” *Functions and Machine Learning*, Nature, 218 (1968), pp. 19–22, <https://doi.org/10.1038/218019a0>.
- [19] R. OSTE AND J. V. DER JEUGT, *Motzkin paths, motzkin polynomials and recurrence relations*, Electronic Journal of Combinatorics, 22 (2015), p. P2.8, <https://doi.org/10.37236/4781>, <https://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i2p8/pdf>.
- [20] D. R. PAGE, *Generalized algorithm for restricted weak composition generation*, Journal of Mathematical Modelling and Algorithms in Operations Research, 12 (2013), pp. 345–372, <https://doi.org/10.1007/s10852-012-9194-4>.
- [21] F. RUSKEY AND A. WILLIAMS, *Generating combinations by prefix shifts*, in Proceedings of the 11th Annual International Computing and Combinatorics Conference (COCOON 2005), vol. 3595 of Lecture Notes in Computer Science, Springer, 2005, pp. 570–576, https://doi.org/10.1007/11533719_58.
- [22] I. STOJMENOVIC, *Recursive algorithms in computer science courses: Fibonacci numbers and*

binomial coefficients, IEEE Transactions on Education, 43 (2000), pp. 273–276, <https://doi.org/10.1109/13.865200>.

- [23] T. TAKAOKA, *Loopless generation of combinatorial objects*, Journal of Computer Science and Technology, 22 (2007), pp. 714–727, <https://doi.org/10.1007/s11390-007-9094-7>.

Appendix E. Excursions. Another useful tool for constructing paths in $\mathcal{P}_{k^*}$ is to exploit *excursions from 0* in the position sequence $\pi(\mathbf{p})$ (see Equation (2.5)).

Motivating example. Assume $D = 9$ and $k^* = 2$. Consider the position sequence

$$(E.1) \quad (0, 1, 2, 1, 0, 1, 0, 0, 1, 2).$$

There are three maximal positive runs (excursions) bounded by zeros:

$$(0 \mid \underbrace{1, 2, 1}_{\text{exc. 1}} \mid 0 \mid \underbrace{1}_{\text{exc. 2}} \mid 0, 0 \mid \underbrace{1, 2}_{\text{exc. 3}}),$$

where the vertical bars mark zeros. Since the last run ends at $d = D$ with value $k^* = 2 \neq 0$, the rightmost block is *locked* (see below) and is not flippable if we wish to preserve the terminal position. Flipping any subset of the *unlocked* excursions (here, excursions 1 and 2) negates the entries inside those runs and yields valid paths that still end at k^* . For instance, flipping excursion 1, excursion 2, both, or neither produces the four sequences

$$(E.2) \quad \begin{aligned} &(0, -1, -2, -1, 0, 1, 0, 0, 1, 2), \\ &(0, 1, 2, 1, 0, -1, 0, 0, 1, 2), \\ &(0, 1, 2, 1, 0, 1, 0, 0, 1, 2), \\ &(0, -1, -2, -1, 0, -1, 0, 0, 1, 2). \end{aligned}$$

Formal setup. Let $\mathbf{k} \stackrel{\text{def}}{=} (k_0, k_1, \dots, k_D) \in \mathbb{Z}^{D+1}$ be a position sequence with $k_0 = 0$ and $k_D = k^*$. We call a pair of indices (l, r) with $0 \leq l < r \leq D + 1$ an *excursion interval* if

$$k_l = 0, \quad k_r = 0 \text{ when } r \leq D, \quad k_d > 0 \text{ for all } l < d < r,$$

and (l, r) is maximal with these properties. (When the final run reaches D with $k_D > 0$, we *close* it by the sentinel $r = D + 1$.) The open index set $(l, r) \cap \{0, 1, \dots, D\}$ is the support of the excursion.

List all excursion intervals from left to right as

$$0 = l_1 < r_1 \leq l_2 < r_2 \leq \dots \leq l_M < r_M \leq D + 1.$$

Define the *excursion indicators* $\mathbf{1}^{(m)} \in \{0, 1\}^{D+1}$ by

$$\mathbf{1}_d^{(m)} = \begin{cases} 1, & l_m < d < r_m, \\ 0, & \text{otherwise.} \end{cases}$$

Set the *lock flag*

$$\epsilon(\mathbf{k}) \stackrel{\text{def}}{=} \begin{cases} 1, & r_M = D + 1 \text{ (rightmost block hits } D \text{ with } k_D \neq 0), \\ 0, & r_M \leq D \text{ (rightmost block ends at a zero).} \end{cases}$$

Thus the indices of *flippable* excursions are

$$\mathcal{I}^*(\mathbf{k}) \stackrel{\text{def}}{=} \{1, 2, \dots, M - \epsilon(\mathbf{k})\}.$$

Flip operator. For any subset $A \subseteq \mathcal{I}^*(\mathbf{k})$ define

$$\mathcal{F}_A(\mathbf{k}) \stackrel{\text{def}}{=} \mathbf{k} - 2 \sum_{m \in A} (\mathbf{k} \odot \mathbf{1}^{(m)}),$$

where $\odot$ denotes the Hadamard (entrywise) product. In words: on each excursion $m \in A$ we negate the entries of $\mathbf{k}$ (equivalently, we reflect the excursion about 0), and we leave all other indices unchanged. The full *flip family* is

$$(E.3) \quad \mathcal{F}(\mathbf{k}) \stackrel{\text{def}}{=} \{\mathcal{F}_A(\mathbf{k}) : A \subseteq \mathcal{I}^*(\mathbf{k})\}.$$

Hence

$$(E.4) \quad |\mathcal{F}(\mathbf{k})| = 2^{M-\epsilon(\mathbf{k})}.$$

If one wishes to exclude the trivial “no-flip” element corresponding to $A = \emptyset$, the count becomes $2^{M-\epsilon(\mathbf{k})} - 1$.

LEMMA E.1 (Validity and endpoint preservation). *Let $\mathbf{k} \in \mathbb{Z}^{D+1}$ be a position sequence with $k_0 = 0$ and $k_D = k^*$, and let $A \subseteq \mathcal{I}^*(\mathbf{k})$. Then $\tilde{\mathbf{k}} \stackrel{\text{def}}{=} \mathcal{F}_A(\mathbf{k})$ is again a valid position sequence of a path in $\mathcal{P}$; i.e., $\tilde{k}_d - \tilde{k}_{d-1} \in \{-1, 0, +1\}$ for all d , with $\tilde{k}_0 = 0$ and $\tilde{k}_D = k^*$.*

Proof. Inside an excursion (l_m, r_m) , the increments of $\mathbf{k}$ are in $\{\pm 1\}$ (because the values are strictly positive and bounded by zeros at the endpoints). Negating the entries on that block negates those increments, which remain in $\{\pm 1\}$. At the boundaries $d = l_m$ and $d = r_m$ we have zeros in both $\mathbf{k}$ and $\tilde{\mathbf{k}}$ (for $r_m \leq D$) or, when $r_m = D + 1$, the block is locked and not flipped by construction. Hence all increments remain in $\{-1, 0, 1\}$. Since each flipped excursion begins and ends at 0, the net displacement contributed by that excursion remains 0, so the terminal value $k_D = k^*$ is preserved (locking prevents altering the final nonzero run). $\square$

DEFINITION E.2 (Nonnegative representatives).

$$\mathcal{P}_{k^*}^+ \stackrel{\text{def}}{=} \{\mathbf{p} \in \mathcal{P}_{k^*} : \pi(\mathbf{p}) \in \mathbb{Z}_+^{D+1}\}.$$

PROPOSITION E.3 (Flip representation). *If $k^* \geq 0$, then*

$$(E.5) \quad \mathcal{P}_{k^*} = \bigcup_{\mathbf{p} \in \mathcal{P}_{k^*}^+} \pi^{-1}(\mathcal{F}(\pi(\mathbf{p}))).$$

If $k^ < 0$, then*

$$(E.6) \quad \mathcal{P}_{k^*} = \bigcup_{\mathbf{p} \in \mathcal{P}_{-k^*}^+} \pi^{-1}(-\mathcal{F}(\pi(\mathbf{p}))).$$

Proof. For $k^* \geq 0$, any $\mathbf{p} \in \mathcal{P}_{k^*}$ has position sequence $\mathbf{k}$ whose negative entries occur in blocks separated by zeros. Successively reflecting each negative block across 0 produces a nonnegative sequence $\mathbf{k}^{(+)} \in \mathbb{Z}_+^{D+1}$ with the same endpoints and increments in $\{-1, 0, 1\}$. Thus $\mathbf{k} \in \mathcal{F}(\mathbf{k}^{(+)})$ with $\mathbf{k}^{(+)} = \pi(\mathbf{p}_+)$ for some $\mathbf{p}_+ \in \mathcal{P}_{k^*}^+$, proving inclusion “ $\subseteq$ ”. The reverse inclusion follows from Lemma E.1. The case $k^* < 0$ reduces to $k^* > 0$ by global sign-flip. $\square$

Counting flips for a fixed representative. Let $\mathbf{k} = \pi(\mathbf{p}) \in \mathbb{Z}_+^{D+1}$ and let M be the number of excursions of $\mathbf{k}$ (maximal positive runs). Then $|\mathcal{F}(\mathbf{k})| = 2^{M-\epsilon(\mathbf{k})}$ by Equation (E.4); when excluding the no-flip element, $|\mathcal{F}(\mathbf{k})| = 2^{M-\epsilon(\mathbf{k})} - 1$. In the example Equation (E.1), $M = 3$ and $\epsilon(\mathbf{k}) = 1$, hence $|\mathcal{F}(\mathbf{k})| = 2^2 = 4$, exactly the four sequences in Equation (E.2).

Appendix F. Step counts and parity. Any $\mathbf{p} \in \mathcal{P}_{k^*}$ can be decomposed into j_+ up-steps, j_- down-steps, and j_0 stays. Then

$$(F.1) \quad j_+ + j_- + j_0 = D, \quad j_+ - j_- = k^*.$$

Solving gives

$$j_+ = \frac{D + k^* - j_0}{2}, \quad j_- = \frac{D - k^* - j_0}{2}.$$

Thus j_+ and j_- are integers iff

$$(F.2) \quad j_0 \equiv D + k^* \pmod{2},$$

and necessarily

$$(F.3) \quad 0 \leq j_0 \leq D, \quad j_+ \leq \left\lfloor \frac{D + k^*}{2} \right\rfloor, \quad j_- \leq \left\lfloor \frac{D - k^*}{2} \right\rfloor.$$

The parity condition Equation (F.2) is the even-odd compatibility between depth D , terminal position k^* , and the number of stays. It is the sole parity restriction induced by the underlying trinary step set, and it is implicitly enforced by our path-generation rules we observed in Section 2 - Section 4. In particular, when seeding the first iteration of the (recursion-free) generator, one must choose j_0 with the parity prescribed by Equation (F.2); then $j_{\pm}$ follow from Equation (F.1).

Integration with the positive-path generator. Algorithm A.3 details a recursion-free enumeration of the nonnegative representatives $\mathcal{P}_{k^*}^+$ that maintains the monotone lexicographic ordering and obeys the rules outlined in Section 2. The full path set $\mathcal{P}_{k^*}$ is obtained by applying $\mathcal{F}$ (Definition Equation (E.3)) to each generated representative, as justified by Proposition E.3. This two-stage procedure *exhausts* all paths ending at k^* without duplication.

Remark F.1 (On ordering). The flip step preserves the terminal index and only toggles signs within excursions bounded by zeros, so it commutes with any ordering that respects the underlying rules and structure introduced in Section 2 - 4. In implementations that “ping-pong” across the path space, one may emit each nonnegative representative as soon as it is generated and then emit its flip family in any deterministic subset order (e.g., lexicographic on $\mathcal{I}^*(\mathbf{k})$), preserving a global total order.

Appendix G. Proof of the Equivalence Relation.

PROPOSITION G.1 (Forward direction: every path has a histogram representation). *For every path $\mathbf{p} \in \mathcal{P}$ there exists a unique cardinality tuple $\tilde{C}(\mathbf{p})$.*

Proof. For a path $\mathbf{p}$ with positions $(k_0, \dots, k_D)$, define the counting measure

$$\nu_{\mathbf{p}} := \sum_{d=0}^D \delta_{k_d}, \quad c_k(\mathbf{p}) := \nu_{\mathbf{p}}(\{k\}).$$

Let

$$k_- := \min_{0 \leq d \leq D} k_d, \quad k_+ := \max_{0 \leq d \leq D} k_d,$$

so the path visits only positions in $[k_-, k_+]$. Then for any test function $v : \mathbb{Z} \rightarrow \mathbb{R}$ we have

$$\sum_{d=0}^D v_{k_d} = \int v \, d\nu_{\mathbf{p}} = \sum_{k=k_-}^{k_+} c_k(\mathbf{p}) v_k.$$

Thus $\hat{C}(\mathbf{p}) = (c_k(\mathbf{p}))_{k=k_-}^{k_+}$ is exactly the (unique) histogram/cardinality tuple of $\mathbf{p}$. By definition of $\mathcal{C}_D := \hat{C}(\mathcal{P})$, every $\hat{c} \in \mathcal{C}_D$ arises from at least one path. $\square$

PROPOSITION G.2 (Reordering operator preserves classes). *Fix $\hat{c} \in \mathcal{C}_D$ and any $\mathbf{p}$ with $\hat{C}(\mathbf{p}) = \hat{c}$. Define a legal reordering operator R on $\mathbf{p}$ to be a bijection of indices $R : \{0, \dots, D\} \rightarrow \{0, \dots, D\}$ such that*

$$\mathbf{p}^R := (k_{R^{-1}(0)}, k_{R^{-1}(1)}, \dots, k_{R^{-1}(D)})$$

is again a valid path in $\mathcal{T}_D$ (that is, $k_{R^{-1}(0)} = 0$ and $|k_{R^{-1}(t)} - k_{R^{-1}(t-1)}| \in \{-1, 0, 1\}$ for all t as written in [Section 2](#)).

LEMMA G.3 (Histogram invariance). *If R is a legal reordering for $\mathbf{p}$, then we have $\hat{C}(\mathbf{p}^R) = \hat{C}(\mathbf{p})$.*

Proof. Reindexing the multiset $\{k_d : 0 \leq d \leq D\}$ by a bijection does not change multiplicities at any level k , hence the histogram counts are preserved. $\square$

PROPOSITION G.4 (Classes are exactly reordering orbits). *For any $\mathbf{p} \in \mathcal{P}$,*

$$\{\mathbf{p}^R : R \text{ legal reordering for } \mathbf{p}\} = \{\mathbf{p}' \in \mathcal{P} : \hat{C}(\mathbf{p}') = \hat{C}(\mathbf{p})\}.$$

Proof. ($\subseteq$) If $\mathbf{p}' = \mathbf{p}^R$ with R a legal reordering, then by the lemma $\hat{C}(\mathbf{p}') = \hat{C}(\mathbf{p})$. ($\supseteq$) Conversely, if $\mathbf{p}'$ has $\hat{C}(\mathbf{p}') = \hat{C}(\mathbf{p})$, then the position sequences (k_d) and (k'_d) have the same multiplicities. Thus there exists a bijection R between time indices matching equal occurrences of each level. By assumption $\mathbf{p}'$ is a valid path, so this R is a legal reordering. Hence $\mathbf{p}' \in \{\mathbf{p}^R\}$. $\square$

COROLLARY G.5 (Partition of the tree). *The sets*

$$[\hat{c}] := \{\mathbf{p} \in \mathcal{P} : \hat{C}(\mathbf{p}) = \hat{c}\}, \quad \hat{c} \in \mathcal{C}_D,$$

are pairwise disjoint and their union equals $\mathcal{P}$. Thus each $\hat{c}$ generates exactly the class of paths with that histogram, and the union of all such classes reconstructs the entire tree without over- or undercounting.

THEOREM G.6 (Minimality of histogram classes). *Let $\sim$ be equality of histograms: $\mathbf{p} \sim \mathbf{p}'$ iff $\hat{C}(\mathbf{p}) = \hat{C}(\mathbf{p}')$. A partition Π of $\mathcal{P}$ is called reordering-invariant if whenever $\mathbf{p} \in B \in \Pi$ and $\mathbf{p}'$ is obtained from $\mathbf{p}$ by a legal reordering operator (as in [Section 2](#)), then $\mathbf{p}' \in B$. Then the histogram partition $\{[\hat{c}]\}_{\hat{c} \in \mathcal{C}_D}$ is the coarsest reordering-invariant partition: every such Π refines $\{[\hat{c}]\}$.*

Proof. Legal reorderings preserve histograms, so each orbit under them is contained in some $[\hat{c}]$. Hence any reordering-invariant partition can only join together whole histogram classes. Therefore no strictly coarser reordering-invariant partition exists. $\square$

THEOREM G.7 (One representative per class covers all vertices and edges). *For each vertex (k, d) with $|k| \leq d \leq D$, let $H(k, d) \in \mathcal{P}$ be the first-hit path that reaches level k for the first time at depth d via the lexicographically minimal feasible prefix, and completes to depth D lexicographically minimally (all moves obeying [Section 2](#)). Write $\hat{c}^{k,d} := \hat{C}(H(k, d))$. Define a selection $R : \mathcal{C}_D \rightarrow \mathcal{P}$ by: for each class $[\hat{c}]$, choose the lexicographically smallest (k, d) with $\hat{c} = \hat{c}^{k,d}$ and set $R(\hat{c}) := H(k, d)$. Then*

$$\bigcup_{\hat{c} \in \mathcal{C}_D} V(R(\hat{c})) = V(\mathcal{T}_D) \quad \text{and} \quad \bigcup_{\hat{c} \in \mathcal{C}_D} E(R(\hat{c})) = E(\mathcal{T}_D).$$

Proof. Vertices For any (k, d) , $H(k, d)$ visits (k, d) by construction; the class $[\hat{c}^{k,d}]$ selects $R(\hat{c}^{k,d}) = H(k, d)$, so $(k, d) \in V(R(\hat{c}^{k,d}))$. As (k, d) was arbitrary, all vertices are covered.

Edges Fix $e = ((k, d-1), (k+s, d))$ with $s \in \{-1, 0, 1\}$. The vertex $(k, d-1)$ is covered above, so $R(\hat{c}^{k,d-1}) = H(k, d-1)$ visits it. In $H(k, d-1)$, the step from depth $d-1$ to d is the lexicographically minimal feasible move out of $(k, d-1)$, realizing one of its incident edges. As $(k, d-1)$ varies over all vertices, each edge of $\mathcal{T}_D$ occurs in some $H(k, d-1)$ and hence in some $R(\hat{c})$. This proves the second equality. $\square$