Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(10): 6141–6175.
DOI: 10.3934/era.2025272
Received: 30 May 2025
Revised: 08 September 2025
Accepted: 10 October 2025
Published: 20 October 2025

Research article

Lattice points of flow polytopes related to caracol graphs

Hua Xin[*]

Institute for Advanced Study in Mathematics, Harbin Institute of Technology, Harbin 150001, China

* **Correspondence:** Email: xinh1028@gmail.com.

Abstract: Flow polytopes are fundamental objects in algebraic combinatorics. In this paper, we study the enumeration of lattice points in flow polytopes associated with (a_1, a_2)-caracol graphs on $n + 2$ vertices. Our main result establishes a closed-form expression for the number of lattice points by constructing an explicit combinatorial bijection between Dyck paths and the integer lattice points of the two-parameter family of polytopes (a_1, a_2)-Car_{n+1}, using pseudo-ladder diagrams together with vector partition techniques. When $a_2 = 1$, the lattice point sequence of caracol polytopes coincides with the OEIS sequence A126216 (The On-Line Encyclopedia of Integer Sequences), which enumerates Schröder paths of semilength n with exactly k peaks. Furthermore, we establish a bijection between Schröder paths and the integer lattice points of the two-parameter family of polytopes (a_1, a_2)-Car_{n+1}. All bijections are implemented as explicit algorithms in Python, with the complete source code provided in the appendix to ensure reproducibility.

Keywords: flow polytopes; caracol graphs; lattice points; lattice paths; Kostant partition functions

1. Introduction

Flow polytopes are classical objects in combinatorial optimization [1], which are certain polytopes associated with acyclic directed graphs with net flow vectors. They have demonstrated intimate connections with several areas of mathematics, such as representation theory [2], diagonal harmonics [3], and toric geometry [4]. The combinatorial and geometric exploration of flow polytopes began with the work of Baldoni and Vergne [5], as well as the unpublished research of Stanley and Postnikov (see [6]). For any integer polytope $\mathcal{P}$, computing the volume of $\mathcal{P}$ and counting the lattice points in $\mathcal{P}$ are two of the fundamental problems. For more detailed insights into estimating the volume and counting the lattice points in polytopes, see [7, 8].

Let G be an acyclic graph on $n + 1$ vertices and m edges with a net flow $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$. The flow polytope $\mathcal{F}_G(\boldsymbol{a}) \subseteq \mathbb{R}^m$ (defined in Section 2) is the set of flows on G with the net flow on its vertices given by $\boldsymbol{a}' := (a_1, \ldots, a_n, -\sum_{i=1}^{n} a_i)$. The number of integer lattice points in $\mathcal{F}_G(\boldsymbol{a})$ is given by a special evaluation of the Kostant partition function, which counts the number of integer-valued flows and is denoted by $K_G(\boldsymbol{a}')$. This function also counts the number of ways to express $\boldsymbol{a}'$ as a sum of the vectors $\boldsymbol{e}_i - \boldsymbol{e}_j$ corresponding to the edges (i, j) with $i < j$ in G. Here and throughout, we assume that the edges are oriented from the smallest to the largest vertex index.

As early as 1984, Lidskii studied the complete graph K_{n+1} in [9], providing important formulas for the number of lattice points and the volume of the associated integer polytope. Related developments on flow polytopes can be found in [10]; see also the special case of the Chan–Robbins–Yuen polytope, resolved in [11]. For arbitrary graphs G with a non-negative integer net flow, Baldoni and Vergne [5] extended Lidskii's result to $\mathcal{F}_G(\boldsymbol{a})$, applying residue techniques to compute their volumes. They called it the Lidskii volume formula, which was later independently obtained by Postnikov and Stanley (unpublished, see [6]). In this paper, we instead focus on the Lidskii formula for lattice points. Mészáros and Morales further investigated $\mathcal{F}_G(a)$ using subdivision methods of the polytopes in [12]. Beyond combinatorial and geometric approaches, there has also been recent interest in analyzing network structures from a dynamic systems perspective. For example, Liu et al. [13] investigated the coherence properties of polygonal networks under noise disturbance, deriving exact analytical solutions and highlighting the impact of network topology on robustness. Although this line of research differs from our combinatorial focus, it illustrates the broader relevance of studying the structural properties of graphs and polytopes across disciplines.

This article is inspired by the work of Benedetti et al. [14], which focuses mainly on volume estimation. In contrast, we adopt purely combinatorial methods to compute the lattice points of flow polytopes associated with the caracol graph, based on pseudo-ladder diagrams and the techniques introduced in [6] to provide a novel interpretation of the Lidskii formula. Baldoni and Vergne generalized Lidskii's result for flow polytopes of an arbitrary graph G. While their formulas are combinatorial in nature, their proofs are based on residue computations. Mészáros and Morales [15] studied flow polytopes and Kostant partition functions for signed graphs, whereas our research specifically targets acyclic directed graphs.

Compared with subdivision techniques, which rely on polytopal subdivisions and the basic reduction rule that introduces additional edges, our approach via pseudo-ladder diagrams avoids altering the edge structure of the caracol graph. Instead, it builds on the classical combinatorial object of Dyck paths, yielding a direct counting interpretation. In this way, pseudo-ladder diagrams provide a purely combinatorial explanation of the Baldoni–Vergne–Lidskii formulas. This approach not only demonstrates the elegance of the Lidskii formula but also elucidates the combinatorial properties encoded in the Baldoni–Vergne–Lidskii formula.

In this paper, we consider the caracol graph Car_{n+2} on $n + 2$ vertices and $3n - 1$ edges (see Figure 1).

- The caracol graph Car_{n+2} is created on $n + 2$ vertices and $3n - 1$ edges, and consists of a path $1 \longrightarrow 2 \longrightarrow 3 \longrightarrow \cdots \longrightarrow n + 1 \longrightarrow n + 2$, and the additional edges are $(2, n + 2), (3, n + 2), \ldots, (n, n + 2)$, and $(1, 3), (1, 4), \ldots, (1, n + 1)$. Caracol graphs were considered by Jang and Kim in [16].

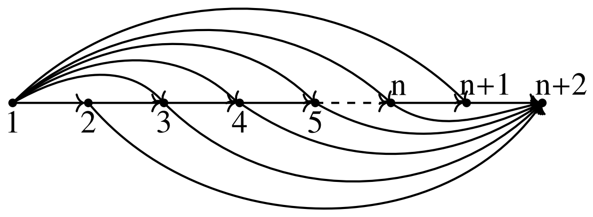


Figure 1. The caracol graph Car_{n+2}.

The volume and the number of lattice points of $\mathcal{F}_G(\boldsymbol{a})$ admit elegant combinatorial formulas for certain graphs G and flow vectors $\boldsymbol{a}$. We first review several known results involving special graphs and the vectors $\boldsymbol{a} = (1, 0, \ldots, 0)$ and $\boldsymbol{a} = (1, 1, \ldots, 1)$. See [14, 17, 18] for additional examples in special cases.

(i) When G is the complete graph K_{n+1} and $\boldsymbol{a} = (1, 0, \ldots, 0)$, $\mathcal{F}_G(\boldsymbol{a})$ is the Chan–Robbins–Yuen polytope [19]. Zeilberger [11] proved that its volume is

$$\operatorname{vol} \mathcal{F}_{K_{n+1}}(1, 0, \ldots, 0) = \prod_{i=1}^{n-2} C_i,$$

where $C_k := \frac{1}{k+1}\binom{2k}{k}$.

(ii) When G is the caracol graph Car_{n+1} and $\boldsymbol{a} = (1, 0, \ldots, 0)$, Mészáros et al. [18] showed that

$$\operatorname{vol} \mathcal{F}_{\mathrm{Car}_{n+1}}(\boldsymbol{a}) = C_{n-2},$$

by identifying the polytope with the order polytope of the poset $[2] \times [n-2]$.

(iii) When G is the complete graph K_{n+1} and $\boldsymbol{a} = (1, 1, \ldots, 1)$, $\mathcal{F}_G(\boldsymbol{a})$ is the Tesler polytope. Meszáros et al. [3] proved that

$$\operatorname{vol} \mathcal{F}_{K_{n+1}}(1, 1, \ldots, 1) = \frac{\binom{n}{2}!}{\prod_{i=1}^{n-2}(2i+1)^{n-i-1}} \cdot \prod_{i=1}^{n-1} C_i.$$

(iv) When G is the zigzag graph Zig_{n+1}, consisting of the path $1 \to 2 \to \cdots \to n+1$ together with edges $(i, i+2)$), and $\boldsymbol{a} = (1, 0, \ldots, 0)$,

$$\operatorname{vol} \mathcal{F}_{\mathrm{Zig}_{n+1}}(1, 0, \ldots, 0) = E_{n-1},$$

where E_{n-1} equals half the number of alternating permutations on $n-1$ letters [20].

(v) When G is the Pitman–Stanley graph PS_{n+1} and $\boldsymbol{a} = (1, 1, \ldots, 1)$, one may view PS_{n+1} as the directed graph obtained from the path $1 \to 2 \to \cdots \to n+1$ by adding edges $(i, n+1)$ for $1 \le i \le n-1$. The flow polytope $\mathcal{F}_{PS_{n+1}}(a)$ is affinely equivalent to the Pitman–Stanley polytope [21]. The volume of lattice points in $\mathcal{F}_{PS_{n+1}}(a)$ is $\operatorname{vol} \mathcal{F}_{PS_{n+1}}(1, 1, \ldots, 1) = n^{n-2}$, and its number is

$$K_{PS_{n+1}}(1, 1, \ldots, 1, -n) = C_n.$$

However, for general netflow vectors on caracol graphs, such as $\boldsymbol{a} = (a_1, a_2, \ldots, a_2)$, no closed-form formula is currently known for the number of lattice points.

Inspired by these previous works, we aim to derive an explicit formula for the number of integer lattice points in the flow polytope $\mathcal{F}_G(\boldsymbol{a})$, where $G = \mathrm{Car}_{n+2}$ and the net flow vector is $\boldsymbol{a} = (a_1, a_2, \ldots, a_2)$. The selection is motivated by earlier studies of the special cases $(a, 0, \ldots, 0)$ and $(a, 1, \ldots, 1)$ and can be regarded as their natural generalization. For convenience, we refer to such a polytope as an (a_1, a_2)-caracol polytope.

To achieve this, we introduce a class of novel combinatorial objects, called pseudo-ladder diagrams, and use $\mathcal{PLD}_G$ to denote the set of all such diagrams associated with G. These diagrams exhibit strong connections to classical combinatorial objects, including (n, m)-Dyck paths and Schröder paths. By enumerating pseudo-ladder diagrams, we provide new combinatorial interpretations of terms in the Lidskii formulas for the lattice points proposed by Baldoni and Vergne [5] in the case of caracol graphs with net flow vectors.

The organization of this article is as follows. Theorem 3.5 establishes that pseudo-ladder diagrams provide a new combinatorial interpretation of $K_G(\boldsymbol{a}')$. Moreover, Theorem 3.6 presents a bijection between Kostant partition functions and (n, m)-Dyck paths, offering a novel combinatorial framework for the lattice points of the (a_1, a_2)-caracol polytope. Finally, Theorem 4.2 shows that the sequence of lattice points for the caracol polytope with the net flow $\boldsymbol{a} = (a_1, 1, \ldots, 1)$ coincides with the sequence A126216, which counts Schröder paths of semilength n with exactly k peaks [22].

2. Background and definitions

The following definitions and results are standard. We include them here for completeness, as they will be used in the next section.

2.1. Flow polytopes

This section reviews the background on flow polytopes and Kostant partition functions, following the exposition in [15].

Let $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ be an integer vector, and let $G = (V(G), E(G))$ be a directed acyclic graph with $n + 1$ vertices labeled by integers and m directed edges. The edge set $E(G)$ consists of ordered pairs (i, j) with $i, j \in V(G)$ and $i < j$.

An $\boldsymbol{a}$-flow on G is a tuple $(b_{ij})_{(i,j)\in E(G)}$ of non-negative real numbers such that for $j = 1, ..., n$,

$$\sum_{(j,k)\in E(G)} b_{jk} - \sum_{(i,j)\in E(G)} b_{ij} = a_j. \tag{2.1}$$

Definition 2.1 (**Flow polytope**). *An $\boldsymbol{a}$-flow on G is defined as an assignment of the flow b_{ij} to each edge $(i, j) \in E(G)$ such that the net flow at each vertex $j \in [n]$ is a_j, and the net flow at the vertex $n + 1$ is $-\sum_{j=1}^{n} a_j$. Let $\mathcal{F}_G(\boldsymbol{a})$ denote the set of all such $\boldsymbol{a}$-flows on G. We regard $\mathcal{F}_G(\boldsymbol{a})$ as a polytope in $\mathbb{R}^m$, called the flow polytope of G with the net flow $\boldsymbol{a}$.*

In this article, both $\boldsymbol{a}$ and $\boldsymbol{a}' = (a_1, \ldots, a_n, -\sum_{i=1}^{n} a_i)$ will be referred to as the net flow vector, depending on the context, since they represent the same concept.

2.2. *Kostant partition function and Lidskii formula*

The Kostant partition function K_G is defined for a vector $\boldsymbol{a} \in \mathbb{Z}^{n+1}$ as follows:

$$K_G(\boldsymbol{a}) = \#\left\{(f(e))_{e\in E(G)} \,\middle|\, \sum_{e\in E(G)} f(e)\,\alpha(e) \;=\; \boldsymbol{a},\; f(e) \in \mathbb{Z}_{\geq 0}\right\}, \tag{2.2}$$

where each $\alpha(e)_{e\in E(G)} \in \mathbb{Z}^n$ is the positive root associated with the edge e of G.

We can also view an $\boldsymbol{a}$-flow on G as a linear combination of the vectors $\boldsymbol{\alpha}_i := \boldsymbol{e}_i - \boldsymbol{e}_{i+1}$ for $1 \leq i \leq n$, which form the set of simple roots of type A_n. For each directed edge $(i, j) \in E(G)$, we associate the vector $(i, j) \mapsto \boldsymbol{e}_i - \boldsymbol{e}_j = \boldsymbol{\alpha}_i + \cdots + \boldsymbol{\alpha}_{j-1}$ and denote the set of such vectors by Ψ^+_G. Note that $\Psi^+_G \subseteq \Psi^+$, which is the set of all positive roots. In this method, Eq (2.3) expresses $\boldsymbol{a}'$ as a linear combination of the vectors $\boldsymbol{\alpha}_i + \cdots + \boldsymbol{\alpha}_{j-1}$

$$\boldsymbol{a}' = \sum_{(i,j)\in E(G)} b_{ij}[\boldsymbol{\alpha}_i + \cdots + \boldsymbol{\alpha}_{j-1}]. \tag{2.3}$$

When the $\boldsymbol{a}$-flow (b_{ij}) is integral, it yields a vector partition of $\boldsymbol{a}'$.

Here, for the polytopes P and Q, we write $P \oplus Q$ for their Minkowski sum, i.e., $P \oplus Q := \{p + q \mid p \in P, q \in Q\}$.

Proposition 2.2 ([5]). *Let G be a graph on the vertex set $V(G) = [n + 1]$, and let $a_1, \dots, a_n$ be non-negative integers. Then*

$$\mathcal{F}_G(\boldsymbol{a}) = a_1\mathcal{F}_G(\boldsymbol{e}_1 - \boldsymbol{e}_{n+1}) \;\oplus\; a_2\mathcal{F}_G(\boldsymbol{e}_2 - \boldsymbol{e}_{n+1}) \;\oplus\; \cdots \;\oplus\; a_n\mathcal{F}_G(\boldsymbol{e}_n - \boldsymbol{e}_{n+1}), \tag{2.4}$$

The Lidskii formula for the Kostant partition function, established by Stanley and Postnikov in their unpublished work [6], provides an explicit expression for the number of lattice points in the flow polytope $\mathcal{F}_G(\boldsymbol{a})$.

Theorem 2.3 (Baldoni-Vergne-Lidskii formulas [5]). *Let G be a directed graph with $n + 1$ vertices and m edges, where each edge is directed from i to j whenever $i < j$, and each vertex $i = 1, \dots, n$ has at least one outgoing edge. Consider the net flow vector $\boldsymbol{a}' = (a_1, \dots, a_n, -\sum_{i=1}^n a_i)$, where $a_i \in \mathbb{Z}_{\geq 0}$ denotes the outflow from vertex i. Define $\mathbf{t} = (t_1, \dots, t_n) := (d_1 - 1, \dots, d_n - 1)$, where d_i is the outdegree of vertex i in G. Then*

$$K_G(\boldsymbol{a}') = \sum_{\boldsymbol{j}} \binom{a_1 + t_1}{j_1} \cdots \binom{a_{n-1} + t_{n-1}}{j_{n-1}} \cdot K_G(j_1 - t_1, \dots, j_{n-1} - t_{n-1}, 0, 0), \tag{2.5}$$

where the sum is taken over all sequences $\boldsymbol{j} = (j_1, \dots, j_n)$ of non-negative integers satisfying $|\boldsymbol{j}| = j_1 + \ldots + j_n = m - n$ and $\sum_{i=1}^k j_i \;\geq\; \sum_{i=1}^k t_i$ for $k = 1, 2, \dots, n$.

2.3. *The lattice path*

The paths we consider in this paper are not allowed to go below the x-axis. We follow the notation in [23] and define the lattice path as follows.

Definition 2.4 (**(n, m)-Dyck path**). *An (n, m)-Dyck path is a lattice path from $(0, 0)$ to (n, m) in the integer lattice $\mathbb{Z}^2$, using steps of the form $(0, 1)$ and $(1, 0)$, and never going below the diagonal line $y = \frac{m}{n}x$. The set of all such paths is denoted by $\mathcal{D}_{n,m}$; see Figure 2(b) for an example.*

When $m = n$, we obtain an ordinary Dyck path of order n; see Figure 2(a). In [24], Du studied the enumeration of (n, m)-Dyck paths with a given number of peaks or humps, providing a combinatorial object that is relevant to our investigation.

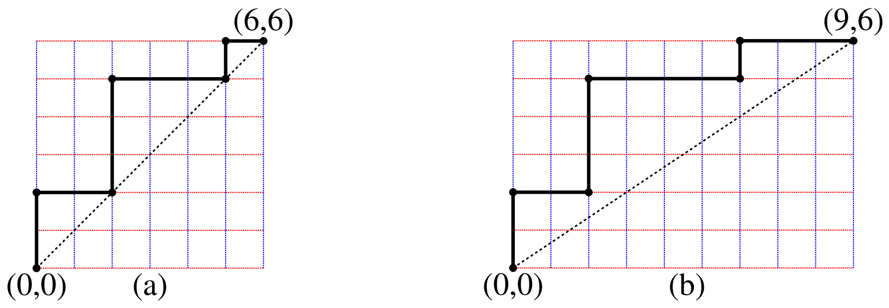


Figure 2. A Dyck path ω and a (9,6)-Dyck path.

Lemma 2.5 ([25, 26]). *The number of Dyck paths of order n with exactly k peaks is*

$$N(n,k) = \frac{1}{n}\binom{n}{k}\binom{n}{k-1}, \tag{2.6}$$

where $N(n,k)$ are the Narayana numbers.

Lemma 2.6 ([24]). *When* $\gcd(n,m) = 1$, *the number of (n,m)-Dyck paths with exactly j peaks is*

$$D(n,m,j) = \frac{1}{j}\binom{n-1}{j-1}\binom{m-1}{j-1}. \tag{2.7}$$

We also note that Eq (2.7) is referred to as the rational Narayana number in [27].

The Narayana number triangle $T(n,k)$, also known as the Catalan triangle, corresponds to the sequence $A001263$ in [22] and is defined as shown below.

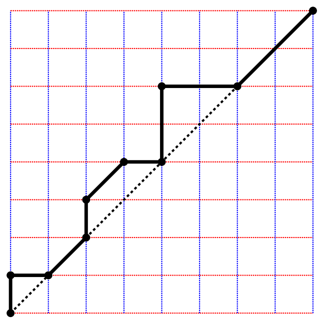
Table 1. The values of the Narayana triangle for $1 \le k \le n$.

n/k	1	2	3	4	5	6	7	8	9
1	1								
2	1	1							
3	1	3	1						
4	1	6	6	1					
5	1	10	20	10	1				
6	1	15	50	50	15	1			
7	1	21	105	175	105	21	1		
8	1	28	196	490	490	196	28	1	
9	1	36	336	1176	1764	1176	336	36	1

A Dyck path is a Schröder path with no diagonal steps $H = (2, 2)$. The study of Dyck and Schröder paths in relation to the Kostant partition function is one of the main focuses of this article.

Definition 2.7 (**Schröder path** [28, 29])**.** *A Schröder path of semilength n is a lattice path from* $(0, 0)$ *to* (n, n) *consisting of north steps* $U = (0, 1)$*, east steps* $D = (1, 0)$*, and diagonal steps* $H = (1, 1)$*, which stays weakly above the diagonal line* $y = x$*. Here, the term* semilength *n refers to the common endpoint* (n, n)*. The set of all Schröder paths of semilength n with k peaks is denoted as* $\mathcal{S}_n^k$*.*

For example, the path $\pi = UDHUHDUUDDHH$, illustrated in Figure 3, is a Schröder path of semilength 8.



$\pi = UDHUHDUUDDHH$

Figure 3. The Schröder path π.

3. A combinatorial interpretation of the (a_1, a_2)-caracol polytope

In this section, we aim to give a combinatorial interpretation of the number of lattice points in the (a_1, a_2)-caracol flow polytope. Our approach relies on introducing a new class of objects, called *pseudo-ladder diagrams*, which allow us to represent the Kostant partition function in a transparent combinatorial form.

3.1. Pseudo-ladder diagrams for the caracol graph

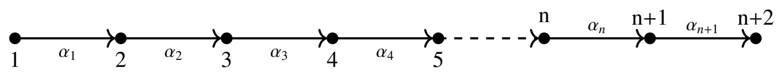
To connect the net flow vector with the vectors α_i, we express $\boldsymbol{c}'$ as a linear combination of the values of α_i.

Given a non-negative integer vector $\boldsymbol{c}' = (c_1, c_2, \ldots, c_{n+1}, -\sum_{i=1}^{n+1} c_i)$, we have

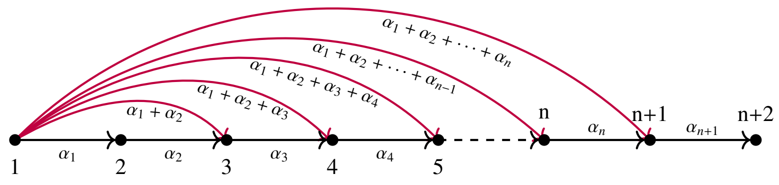
$$\begin{aligned}
\boldsymbol{c}' &= c_1e_1 + c_2e_2 + \cdots + c_{n+1}e_{n+1} - (c_1 + c_2 + \cdots + c_{n+1})e_{n+2}\\
&= c_1(e_1 - e_2 + e_2) + c_2(e_2 - e_3 + e_3) + \cdots + c_{n+1}(e_{n+1} - e_{n+2} + e_{n+2}) - (c_1 + \cdots + c_{n+1})e_{n+2}\\
&= c_1(e_1 - e_2) + c_2(e_2 - e_3) + \cdots + c_{n+1}(e_{n+1} - e_{n+2}) + \sum_{k=1}^{n+1} c_k e_{k+1} - (c_1 + c_2 + \cdots + c_{n+1})e_{n+2}\\
&= \sum_{i=1}^{n+1} c_i\alpha_i + \sum_{j=1}^{n} c_j\alpha_{j+1} + \sum_{k=1}^{n-1} c_k e_{k+2} - (c_1 + c_2 + \cdots + c_{n-1})e_{n+2}\\
&= c_1\alpha_1 + (c_1 + c_2)\alpha_2 + \cdots + (c_1 + c_2 \cdots + c_{n+1})\alpha_{n+1}.
\end{aligned}$$

We can rewrite $\boldsymbol{c}'$ as a linear combination $\boldsymbol{c}' = c_1\alpha_1 + (c_1 + c_2)\alpha_2 + \cdots + (c_1 + \cdots + c_{n+1})\alpha_{n+1}$, where each c_i is a non-negative integer for $1 \le i \le n+1$. Our goal is to compute $K_{\mathrm{Car}_{n+2}}(\boldsymbol{c}')$ for the $\boldsymbol{c}'$ above. Similarly, a non-negative integer net flow vector $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1 + na_2))$ can be expressed as $\boldsymbol{a}' = a_1\alpha_1 + (a_1 + a_2)\alpha_2 + \cdots + (a_1 + na_2)\alpha_{n+1}$.

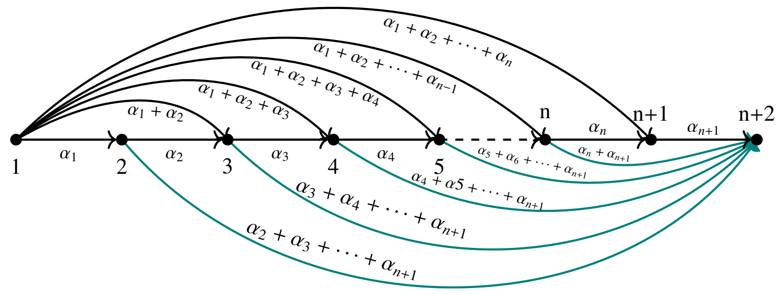
Figure 4 illustrates the caracol graph Car_{n+2}. We then classify $\boldsymbol{c}'$ into the following three distinct forms.



$[\alpha_i]$, for all $1 \le i \le n+1$


$\alpha_1 + \alpha_2 + \cdots + \alpha_n$
$\alpha_1 + \alpha_2 + \cdots + \alpha_{n-1}$
$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$
$\alpha_1 + \alpha_2 + \alpha_3$
$\alpha_1 + \alpha_2$
1
α_1
2
α_2
3
α_3
4
α_4
5
n
α_n
n+1
α_{n+1}
n+2

$[\alpha_1 + \alpha_2 + \cdots + \alpha_i]$, for all $2 \le i \le n$


$\alpha_1 + \alpha_2 + \cdots + \alpha_n$
$\alpha_1 + \alpha_2 + \cdots + \alpha_{n-1}$
$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4$
$\alpha_1 + \alpha_2 + \alpha_3$
$\alpha_1 + \alpha_2$
1
α_1
2
α_2
3
α_3
4
α_4
5
n
α_n
n+1
α_{n+1}
n+2
$\alpha_n + \alpha_{n+1}$
$\alpha_5 + \alpha_6 + \cdots + \alpha_{n+1}$
$\alpha_4 + \alpha 5 + \cdots + \alpha_{n+1}$
$\alpha_3 + \alpha_4 + \cdots + \alpha_{n+1}$
$\alpha_2 + \alpha_3 + \cdots + \alpha_{n+1}$

$[\alpha_i + \alpha_{i+1} + \cdots + \alpha_{n+1}]$, for all $2 \le i \le n$

Figure 4. The caracol graph Car_{n+2} with three types of labeled edges.

Motivated by the unique structure of the net flow vector in flow polytopes of (a_1, a_2)-caracol graphs, we introduce the notion of *pseudo-ladder diagrams*. This concept provides a graphical way to encode the vector partitions of the net flow vector, thereby bridging algebraic decompositions with combinatorial representations.

Definition 3.1 (**Pseudo-ladder diagrams**). *A pseudo-ladder diagram provides a combinatorial interpretation of the Kostant partition function $K_{\mathrm{Car}}(\boldsymbol{c}')$ through the following algorithm.*

The construction of pseudo-ladder diagrams follows a sequence of algebraic and diagrammatic steps. In particular, Steps 1–3 describe the algebraic decomposition, while Steps 4–7 translate the result into a graphical representation.

Algorithm 1: Enumeration algorithm for pseudo-ladder diagrams

```
Input:
  ① A net flow vector $\boldsymbol{c}' = (c_1, c_2, \dots, c_{n+1})$;
  ② $p$ is used to represent different vector partitions.
Output: Pseudo-ladder diagrams $\mathcal{PLD}_G(\boldsymbol{c}')$.
Step 1: Input a net flow vector $\boldsymbol{c}' = (c_1, c_2, \dots, c_{n+1})$;
Step 2: Specify three different forms of vector partitions;
for $i \leftarrow 1$ to $n + 1$ do
    $p \leftarrow [\alpha_i]$;
for $i \leftarrow 2$ to $n$ do
    $p \leftarrow [\alpha_1 + \alpha_2 + \cdots + \alpha_i]$;
for $i \leftarrow 2$ to $n$ do
    $p \leftarrow [\alpha_i + \alpha_{i+1} + \cdots + \alpha_{n+1}]$;
Step 3: Express the net flow vector as an integer vector partition:
    $$\boldsymbol{c}' = c_1\alpha_1 + (c_1 + c_2)\alpha_2 + \cdots + (c_1 + \cdots + c_{n+1})\alpha_{n+1}.$$
Step 4: for $i \leftarrow 1$ to $n + 1$ do
    coefficient of $\alpha_i \leftarrow$ number of points on each row;
Step 5: Rules for the vector partition $[\alpha_i]$ of an individual edge:
    1. Solid gray points represent the vector partition $[\alpha_i]$ of an individual edge;
    2. Vector partitions of $\alpha_i$ at different places in the same line are considered equivalent.
Step 6: The purple line segment represents the vector partition $[\alpha_1 + \alpha_2 + \cdots + \alpha_i]$;
Step 7: The teal line segment represents the vector partition $[\alpha_i + \alpha_{i+1} + \cdots + \alpha_{n+1}]$;
Step 8: If all rules are satisfied, then output $\mathcal{PLD}_G(\boldsymbol{c}')$;
Notes:
    1. Gray points are arranged top to bottom according to $\alpha_1, \alpha_2, \dots, \alpha_{n+1}$.
    2. Gray points of each row are arranged from left to right.
    3. All line segments are placed from left to right.
    4. All objects are arranged from left to right (see Example 3.2).
    5. The calculation process for the number of vector partitions is illustrated in Appendix 1.
```

Applying Algorithm 1 to the caracol graph with the net flow vector $\boldsymbol{c}' = (3, 1, \dots, 1, -7)$, we obtain the following two pseudo-ladder diagrams corresponding to the vector $3\alpha_1 + 4\alpha_2 + \cdots + 7\alpha_5$.

Example 3.2. *Consider the caracol graph with the net flow* $\boldsymbol{c}' = (3, 1, \ldots, 1, -7)$. *We consider two pseudo-ladder diagrams corresponding to the vector* $3\alpha_1 + 4\alpha_2 + 5\alpha_3 + 6\alpha_4 + 7\alpha_5$ *as follows:*

① $2[\alpha_1] + [\alpha_2] + 2[\alpha_5] + [\alpha_4 + \alpha_5] + 2[\alpha_3 + \alpha_4 + \alpha_5] + [\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4] + 2[\alpha_2 + \alpha_3 + \alpha_4 + \alpha_5]$;

② $2[\alpha_1] + [\alpha_2] + 2[\alpha_3] + [\alpha_4] + 2[\alpha_5] + [\alpha_1 + \alpha_2] + 2[\alpha_4 + \alpha_5] + [\alpha_3 + \alpha_4 + \alpha_5] + 2[\alpha_2 + \alpha_3 + \alpha_4 + \alpha_5]$.

The corresponding pseudo-ladder diagrams are depicted in Figure 5.

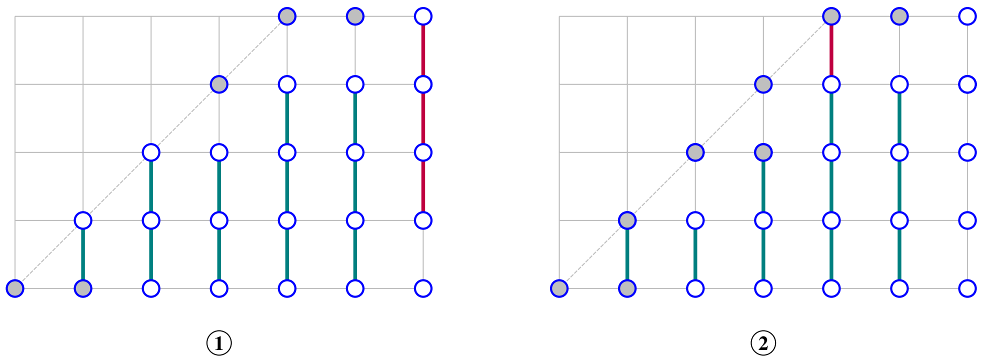


Figure 5. Pseudo-ladder diagrams corresponding to the vector $3\alpha_1 + 4\alpha_2 + 5\alpha_3 + 6\alpha_4 + 7\alpha_5$.

Remark 3.3. *As illustrated in Example 3.2, different pseudo-ladder diagrams may correspond to the same vector partition. This observation motivates the following notion of equivalence. There can be multiple pseudo-ladder diagrams corresponding to a given net flow vector. For example, the vector partition of the form* $[\alpha_i + \alpha_{i+1} + \cdots + \alpha_{n+1}]$ *for* $1 \le i \le n$ *is equivalent to the partition* $[\alpha_i + \alpha_{i+1} + \cdots + \alpha_j] + [\alpha_j + \alpha_{j+1} + \cdots + \alpha_{n+1}]$ *for any* j *with* $i \le j \le n$.

Two pseudo-ladder diagrams are called equivalent if they represent the same vector partition of $\boldsymbol{c}'$. *Let* $\mathcal{PLD}_G(\boldsymbol{c}')$ *denote the set of pseudo-ladder diagrams corresponding to all vector partitions of* $\boldsymbol{c}'$, *and* $|\mathcal{PLD}_G(\boldsymbol{c}')|$ *denotes its cardinality.*

We illustrate the concepts above with the smallest nontrivial case of the caracol graph, showing all pseudo-ladder diagrams corresponding to a given net flow vector.

Example 3.4. *In this example, we consider the smallest nontrivial case of the caracol graph, namely the case with* $n + 2 = 4$ *vertices. For the net flow vector* $\boldsymbol{a}' = (2, 1, 1, -4)$, *a direct computation shows that* $K_{\mathrm{Car}_4}(2, 1, 1, -4) = 9$.

According to the splitting rules established in this paper, each integer vector partition corresponds bijectively to a pseudo-ladder diagram, and vice versa. For illustration, we display below the nine pseudo-ladder diagrams arising from this correspondence (Figure 6).

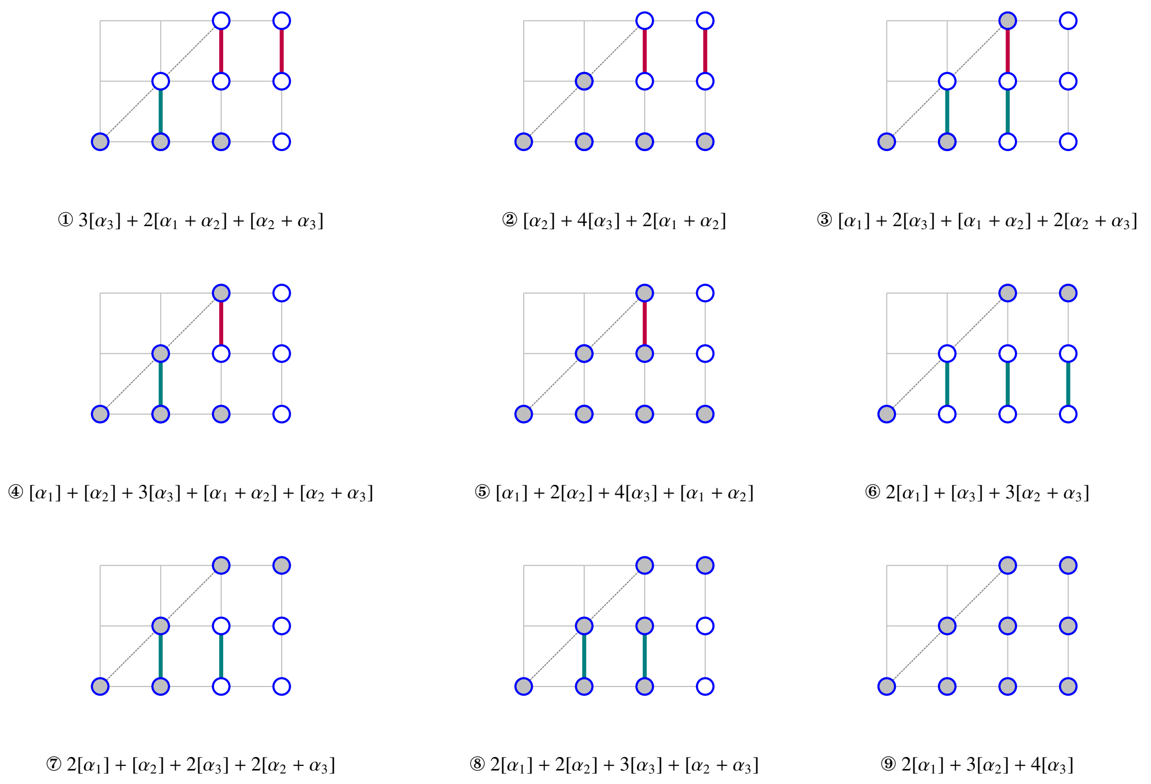

① 3[α3] + 2[α1 + α2] + [α2 + α3]
② [α2] + 4[α3] + 2[α1 + α2]
③ [α1] + 2[α3] + [α1 + α2] + 2[α2 + α3]
④ [α1] + [α2] + 3[α3] + [α1 + α2] + [α2 + α3]
⑤ [α1] + 2[α2] + 4[α3] + [α1 + α2]
⑥ 2[α1] + [α3] + 3[α2 + α3]
⑦ 2[α1] + [α2] + 2[α3] + 2[α2 + α3]
⑧ 2[α1] + 2[α2] + 3[α3] + [α2 + α3]
⑨ 2[α1] + 3[α2] + 4[α3]

Figure 6. The pseudo-ladder diagrams corresponding to the specified net flow vector.

From the construction described above, we have the following result.

Theorem 3.5. *For any caracol graph on* $n+2$ *vertices, and let* $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1 + na_2)) \in \mathbb{Z}^{n+2}$ *be a non-negative net flow vector. Then,*

$$K_G(\boldsymbol{a}') = |\mathcal{PLD}_G(\boldsymbol{a}')|\,. \tag{3.1}$$

3.2. The main result

On the basis of the construction described above, we arrive at the following result.

Theorem 3.6. *Let* a_1 *and* a_2 *be positive integers, and let* $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1 + na_2))$ *be a non-negative integer net flow vector. Then the lattice points of the flow polytope* $\mathcal{F}_{\mathrm{Car}_{n+2}}(\boldsymbol{a}')$ *can be computed as follows:*

$$K_{\mathrm{Car}_{n+2}}(a_1, a_2, \ldots, a_2, -(a_1 + na_2)) = \frac{1}{n}\binom{a_1 + n - 1}{n-1}\binom{a_1 + n(a_2+1)}{n-1}, \tag{3.2}$$

where $n > 0$*. This expression is equivalent to the number of* $(\tilde{n}, m)$*-Dyck paths with* j *peaks:*

$$D(\tilde{n}, m, j) = \frac{1}{j}\binom{n + a_1 - 1}{j-1}\binom{m-1}{j-1}, \tag{3.3}$$

provided that $\gcd(\tilde{n}, m) = 1$*. Here,* $\tilde{n} = n + a_1$*,* $j = n$*, and* $m = a_1 + na_2 + n + 1$*.*

Proof. We prove Eq (3.3) by constructing an intriguing visual bijection between the pseudo-ladder diagrams $\mathcal{PLD}_G$ and the set of Dyck paths $\mathcal{D}_{n,m}$ from $(0,0)$ to $(a_1+n, a_1+n(a_2+1)+1)$ that never go above the diagonal line $y = \frac{a_1+n(a_2+1)+1}{a_1+n}x$. This bijection is defined as $\varphi\colon \mathcal{PLD}_G \to \mathcal{D}_{n,m}$.

If $\rho \in \mathcal{PLD}_G$ represents a pseudo-ladder diagram for non-negative net flow $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1 + na_2))$, $\varphi(\rho)$ is obtained by traversing ρ from the starting column to the final column and applying the following algorithm for each column.

```
Algorithm 2: Transforming pseudo-ladder diagrams $\mathcal{PLD}_G(\boldsymbol{a}')$ into Dyck paths $\mathcal{D}_n^k$

  Input: Pseudo-ladder diagrams $\mathcal{PLD}_G(\boldsymbol{a}')$
  Output: A Dyck path $\mathcal{D}_n^k$
    1. Start from $(0,0)$ and move right along the horizontal axis using level steps of the form $(1,0)$.
    2. In the pseudo-ladder diagrams $\mathcal{PLD}_G(\boldsymbol{a}')$:
         ① If a horizontal step intersects a vertical step of the diagram;
         ② Move to the upper vertex of the vertical step along the straight line;
         ③ Take a level step at this position to proceed to the next column.
    3. After reaching the end of the rightmost column of the diagram:
         ① Move one level step;
         ② Then take $a_1 + na_2$ vertical steps to reach $(a_1 + n,\ a_1 + na_2 + n + 1)$.
    4. For $1 \le i \le n+1$, only pass through the solid gray points in the segment vector partitions.
    5. If all the conditions are satisfied, then output the Dyck path $\mathcal{D}_n^k$;
  Note: The number of peaks is $k$.
```

Note that each pseudo-ladder diagram of the integer net flow $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1 + na_2))$ corresponds to a Dyck path. Thus, we obtain an (a_1+n, a_1+na_2+n+1)-Dyck path. Each step of this mapping is clearly reversible. Specifically, given an (n,m)-Dyck path γ, the inverse mapping $\varphi^{-1}(\gamma)$ can be constructed as follows:

(i) Rotate the Dyck path γ 180 degrees.

(ii) Remove all east steps as well as the last north step.

(iii) Determine the number of points in each row, denoted $\alpha_1, \alpha_2, \ldots, \alpha_i$, from top to bottom, where $1 \le i \le n+1$; align the points in each row to the right.

(iv) Locate the last north step in the $(n+2)$-th column; copy this step downward by $n-2$ units until it reaches the line corresponding to α_n, and color the resulting line segment purple.

(v) Add a line segment to the north step in the first $n+1$ columns, extending it to the line α_{n+1}; if there is no north step between two adjacent columns, add a line segment of the same height as the previous column, ensuring its endpoint lies on the line α_{n+1}.

(vi) Write the vector partition based on the line segment in each column, then add the remaining points from left to right in each row according to the vector $\boldsymbol{c}' := c_1\alpha_1+(c_1+c_2)\alpha_2+\cdots+(c_1+\cdots+c_{n+1})\alpha_{n+1}$.

Then a pseudo-ladder diagram $\mathcal{PLD}_G$ can be derived from a Dyck path $\mathcal{D}_{n,m}$, allowing us to determine the corresponding non-negative net flow $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1+na_2))$. Figure 7 illustrates the mapping $\varphi : \mathcal{PLD}_G \to \mathcal{D}_{n,m}$. Details of the algorithm's implementation are provided in Appendix 2. □

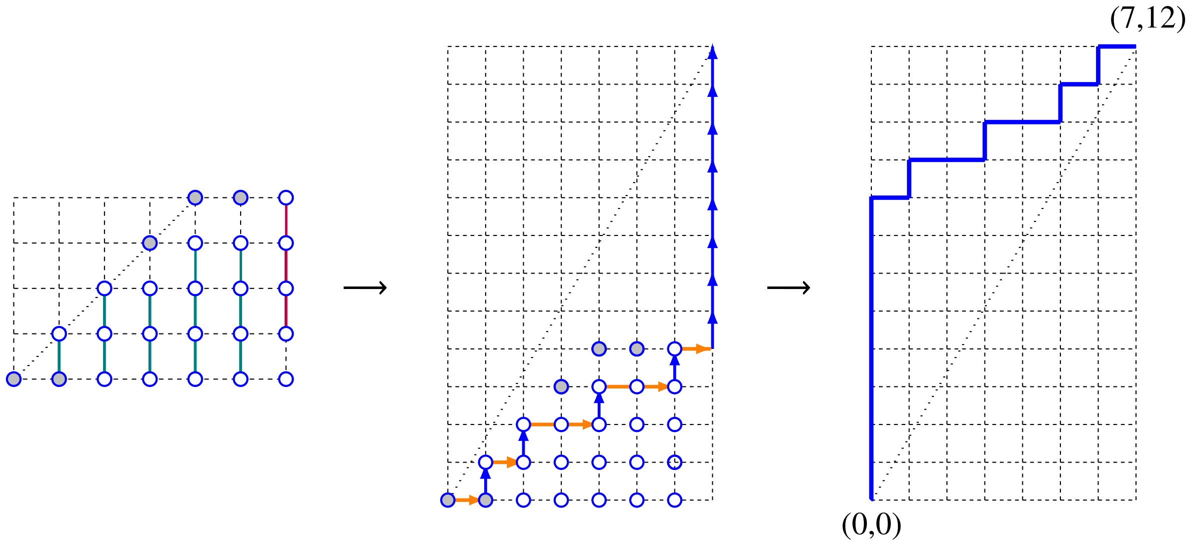


Figure 7. Mapping between the pseudo-ladder diagram of $(Car_6)|_5$ and the free Dyck path in Theorem 3.4.

4. A special case of the $(a_1, 1)$-caracol polytope involving Schröder paths

In this section, we focus on the special case where $a_2 = 1$ in the net flow $\boldsymbol{a}' = (a_1, a_2, \ldots, a_2, -(a_1+na_2))$. We provide a combinatorial interpretation of the caracol polytope with the net flow $\boldsymbol{a}' = (a_1, 1, \ldots, 1, -(a_1+n))$. To this end, we construct a bijection between Kostant partition functions and Schröder paths, which leads to the following theorem.

Notably, when $a_2 = 1$, the number of lattice point sequences in the flow polytopes associated with the caracol graphs corresponds to the sequence A126216 in [22]. Let $S(n, k)$ denote the number of Schröder paths of semilength n with exactly k peaks, but no peaks at level one ($n \geq 1$, $0 \leq k \leq n-1$). See Tables 2 and 3 for more details.

Table 2. The values of $S(n,k)$ for $n \geq 1$ and $0 \leq k \leq n-1$.

$S(n,k)$ \ k / n	0	1	2	3	4	5	6	7	8
1	1								
2	2	1							
3	5	5	1						
4	14	21	9	1					
5	42	84	56	14	1				
6	132	330	300	120	20	1			
7	429	1287	1485	825	225	27	1		
8	1430	5005	7007	5005	1925	385	35	1	
9	4862	19448	32032	28028	14014	4004	616	44	1

Lemma 4.1 ([22]). *The number of Schröder paths of semilength n with exactly k peaks is*

$$S(n,k) = \frac{1}{n}\binom{n}{k}\binom{2n-k}{n+1}. \tag{4.1}$$

For convenience, by substituting n with $n-2$ in Eq (3.2), we obtain the following result for the caracol graph with n vertices.

Theorem 4.2. *Let a_1 and a_2 be positive integers. For the (a_1,a_2)-caracol graph, when $a_2 = 1$, the number of lattice points in the flow polytope $\mathcal{F}_{\mathrm{Car}_n}(a_1,1,1,\ldots,1)$ is given by*

$$K_{\mathrm{Car}_n}(a_1,1,1,\ldots,1,-(a_1+n-2)) = \frac{1}{n-2+a_1}\binom{n-2+a_1}{a_1}\binom{2n-4+a_1}{n-1+a_1}, \tag{4.2}$$

where $n > 2$. This formula counts the number of $(n-2+a_1,a_1)$-Schröder paths of semilength $n-2+a_1$ with exactly a_1 peaks and no peaks at height 0. We denote this number by $T(n-2+a_1,a_1)$. Hence,

$$S(n-2+a_1,a_1) = T(n-2+a_1,a_1). \tag{4.3}$$

Proof. We observe that

$$K_{\mathrm{Car}_n}(a_1,a_2,a_2,\ldots,a_2,-(a_1+(n-2)a_2)) = \frac{1}{n-2}\binom{n-3+a_1}{n-3}\binom{a_1+(a_2+1)(n-2)}{n-3}, \quad n > 2.$$

By setting $a_2 = 1$, we obtain

$$K_{\mathrm{Car}_n}(a_1,1,1,\ldots,1,-(a_1+n-2)) = \frac{1}{n-2}\binom{n-3+a_1}{n-3}\binom{a_1+2(n-2)}{n-3}$$

$$
\begin{aligned}
&= \frac{1}{n-2}\binom{n-3+a_1}{n-3}\binom{2n-4+a_1}{n-3}\\
&= \frac{1}{n-2}\binom{n-3+a_1}{n-3}\binom{2n-4+a_1}{n-1+a_1}\\
&= \frac{1}{n-2+a_1}\binom{n-2+a_1}{a_1}\binom{2n-4+a_1}{n-1+a_1}\\
&= T(n-2+a_1, a_1).
\end{aligned}
$$

Thus, the expression agrees with the enumerative interpretation in terms of Schröder paths, as claimed.

Table 3. The values of the $K_{\mathrm{Car}_n}(a_1, 1, 1, \ldots, 1, -(a_1 + n - 2))$ for $n > 2$.

K_{Car_n} $\backslash$ n / $\boldsymbol{a'}$	2	3	4	5	6	7	8	9
$(0, 1, \ldots, 1)$	1	2	5	14	42	132	429	1430
$(1, 1, \ldots, 1)$		1	5	21	84	330	1287	5005
$(2, 1, \ldots, 1)$		1	9	56	300	1485	7007	32032
$(3, 1, \ldots, 1)$		1	14	120	825	5005	28028	148512
$(4, 1, \ldots, 1)$		1	20	225	1925	14014	91728	556920
$(5, 1, \ldots, 1)$		1	27	385	4004	34398	259896	1790712

Furthermore, we establish a bijection ϕ between the pseudo-ladder diagrams $\mathcal{PLD}_G$ and the set of Schröder paths $\mathcal{S}_n^k$.

For a given pseudo-ladder diagram ρ, the corresponding Schröder path $\phi(\rho)$ can be obtained using the following algorithm.

Algorithm 3: Converting pseudo-ladder diagrams $\mathcal{PLD}_G(\boldsymbol{a}')$ into Schröder paths $\mathcal{S}_n^k$

```
Input: A pseudo-ladder diagram $\mathcal{PLD}_G(\boldsymbol{a}')$
Output: A Schröder path $\mathcal{S}_n^k$ (semilength $n$)
Step 1: Start at $(0,0)$ and treat each level step as an east step $(1,0)$ moving from left to right;
Step 2: Traverse the pseudo-ladder diagram $\mathcal{PLD}_G(\boldsymbol{a}')$ as follows:
    ① Treat each vertical step as a north step $(0,1)$;
    ② If a level step intersects a vertical step, follow the vertical step to its endpoint;
    ③ Then take a level step to the next column;
    ④ Repeat this process;
    ⑤ Continue until reaching the rightmost endpoint in the final column of $\mathcal{PLD}_G(\boldsymbol{a}')$;
Step 3: Remove all solid gray points from the diagram;
Step 4: Replace every pair of consecutive level steps with a level step followed by a diagonal step $(2,2)$;
Step 5: Add a level step and then a north step to return to the main diagonal $y = x$;
Step 6:
    ① Rotate the entire path by $180°$;
    ② Remove any lines that are not part of the final lattice path to obtain the desired Schröder path;
Step 7: If all the conditions are satisfied, then output $\mathcal{S}_n^k$;
Note: The number of semilength is $n$.
```

It has been established that the desired Schröder path of semilength $n-2+a_1$ can be obtained via the construction described above. Moreover, the mapping between pseudo-ladder diagrams and Schröder paths is bijective, allowing the reconstruction of a corresponding pseudo-ladder diagram from any given Schröder path. Figure 8 illustrates this mapping process. The details of the algorithm's implementation are provided in Appendix 2.

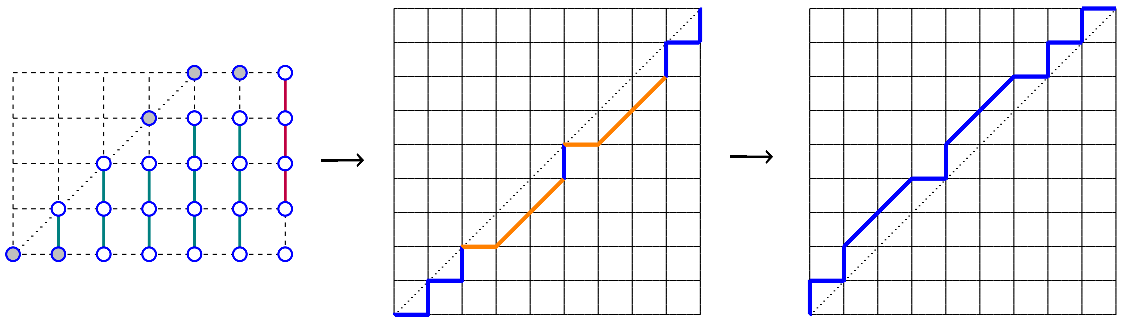


Figure 8. A pseudo-ladder diagram and its corresponding Schröder path.

□

Example 4.3. *Consider the caracol graph with* 6 *vertices shown in Figure* 9*, where* $a_1 = 3$*,* $a_2 = 1$*, and* $\boldsymbol{a}' = (3, 1, \ldots, 1, -7)$*. In this case, we perform the following computation:*

$$\begin{aligned}
D(\tilde{n}, m; j) &= \frac{1}{j}\binom{\tilde{n}-1}{j-1}\binom{m-1}{j-1} \\
&= \frac{1}{n}\binom{a_1+n-1}{j-1}\binom{a_1+na_2+n-1}{j-1} \\
&= \frac{1}{4}\binom{7-1}{4-1}\binom{12-1}{4-1} \\
&= D(7, 12; 4) \\
&= \frac{1}{4}\binom{6}{3}\binom{11}{3} \\
&= 825.
\end{aligned}$$

Simultaneously, the number of lattice points in $\mathcal{F}_{\mathrm{Car}_{n+2}}(a_1, a_2, \ldots, a_2, -(a_1 + na_2))$ *is given by*

$$\begin{aligned}
K_{\mathrm{Car}_{n+2}}(a_1, a_2, \ldots, a_2, -(a_1 + na_2)) &= \frac{1}{n}\binom{a_1+n-1}{n-1}\binom{a_1+na_2+n}{n-1} \\
&= \frac{1}{4}\binom{a_1+3}{3}\binom{a_1+4a_2+4}{3} \\
&= \frac{1}{4}\binom{3+3}{3}\binom{3+4+4}{3} \\
&= K_{\mathrm{Car}_6}(3, 1, 1, 1, 1, -7) \\
&= 825.
\end{aligned}$$

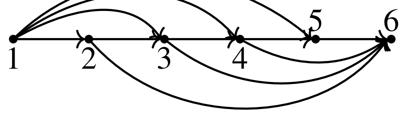


Figure 9. The caracol graph with 6 vertices.

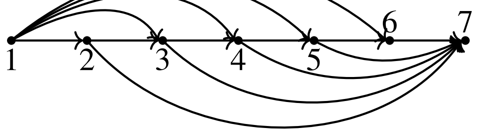


Figure 10. The caracol graph with 7 vertices.

Example 4.4. *Consider the caracol graph with* 7 *vertices shown in Figure* 10*, where* $a_1 = 2$*,* $a_2 = 1$*, and* $\boldsymbol{c}' = (2, 1, \ldots, 1, -7)$*. In this case, we have the following calculation:*

$$\begin{aligned}
D(\tilde{n}, m; j) &= \frac{1}{j}\binom{\tilde{n}-1}{j-1}\binom{m-1}{j-1} \\
&= \frac{1}{n}\binom{a_1+n-1}{j-1}\binom{a_1+na_2+n-1}{j-1}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{5}\binom{7-1}{5-1}\binom{13-1}{5-1}\\
&= D(7,13;5)\\
&= \frac{1}{5}\binom{6}{4}\binom{12}{4}\\
&= 1485.
\end{aligned}
$$

Simultaneously, the number of lattice points in $\mathcal{F}_{\mathrm{Car}_{n+2}}(a_1, a_2, \ldots, a_2, -(a_1 + na_2))$ *is given by*

$$
\begin{aligned}
K_{\mathrm{Car}_{n+2}}(a_1, a_2, \ldots, a_2, -(a_1 + na_2)) &= \frac{1}{n}\binom{a_1+n-1}{n-1}\binom{a_1+na_2+n}{n-1}\\
&= \frac{1}{5}\binom{a_1+4}{4}\binom{a_1+5a_2+5}{4}\\
&= \frac{1}{5}\binom{2+4}{4}\binom{2+5+5}{4}\\
&= K_{\mathrm{Car}_7}(2,1,\ldots,1,-7)\\
&= 1485.
\end{aligned}
$$

Conjecture 4.5. *For positive integers* a_1, a_2, *and* a_3, *let* $\boldsymbol{a}' = (a_1, a_2, a_3, \ldots, a_3, -(a_1 + a_2 + (n-1)a_3))$ *be a net flow vector with non-negative entries, where* $n \geq 1$. *The number of lattice points of the flow polytope* $\mathcal{F}_{\mathrm{Car}_{n+2}}(\boldsymbol{a}')$ *is given by*

$$
K_{\mathrm{Car}_{n+2}}(\boldsymbol{a}') = \frac{a_1 + na_2 + n}{(n-1)n}\binom{a_1+n-1}{n-1}\binom{a_1+a_2+(n-1)(a_3+1)}{n-2}. \tag{4.4}
$$

The conjecture above is based on mathematical experiments conducted in Maple, where the values of $K(\boldsymbol{a}')$ were computed for various values of $\boldsymbol{a}'$, revealing consistent patterns. The proof of this conjecture would also yield new proofs of Theorems 3.6 and 4.2.

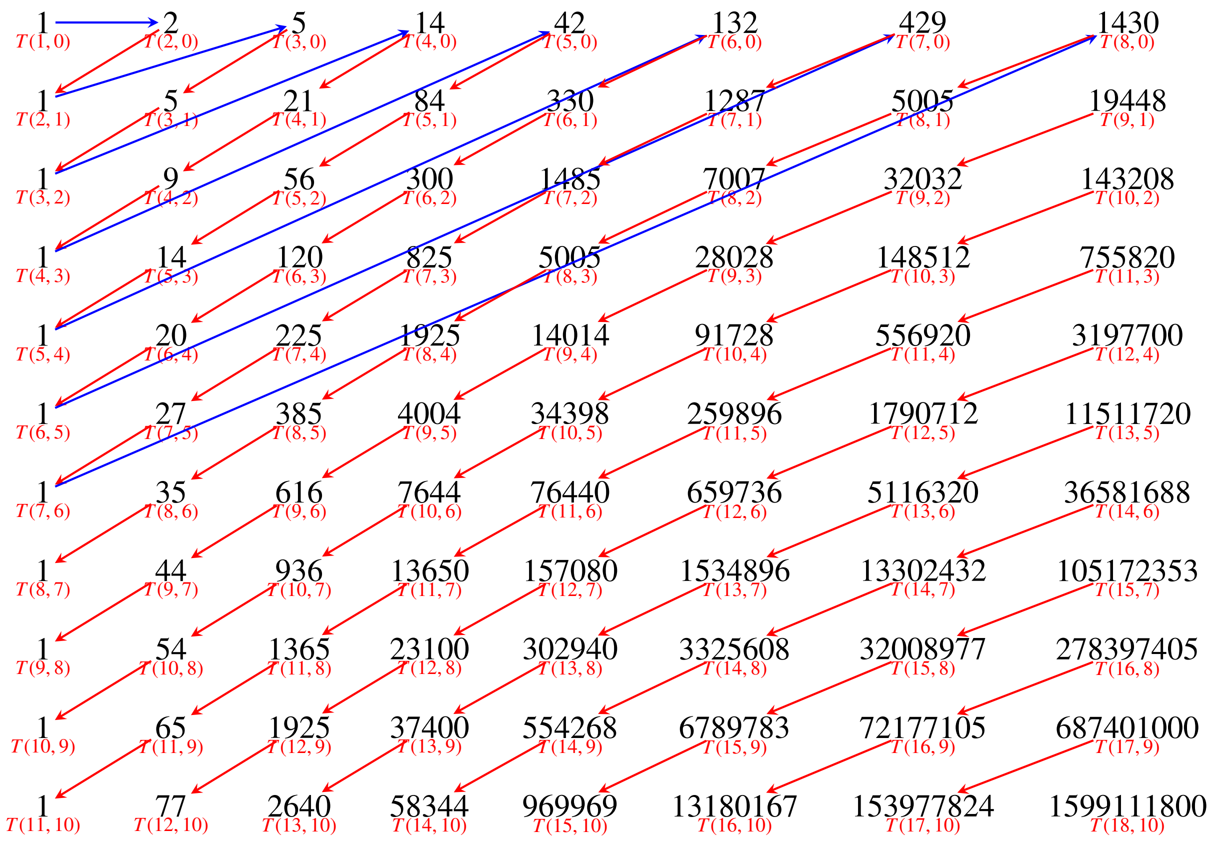
5. Discussion

In this section, we provide several interpretations and consequences of our main results. First, we observe that the enumeration formulas obtained in Theorem 4.2 coincide with certain integer sequences recorded in the OEIS.

• When $a_2 = 1$, we calculated the values of $K_{Car_n}(a)$ for different values of n and obtained some interesting sequences. Moreover, we discovered that the sequence A126216 can be obtained in the following way, which is precisely the number of Schröder paths of semilength n containing exactly k peaks but no peaks at level one. Thus, we established a bijection between caracol graphs and Schröder paths.

Table 4. Enumeration of $K_{\mathrm{Car}_n}(a)$ for vectors of the form $(a_1, 1, \ldots, 1)$.

Vector (a)	$K_{\mathrm{Car}_3}(a)$	$K_{\mathrm{Car}_4}(a)$	$K_{\mathrm{Car}_5}(a)$	$K_{\mathrm{Car}_6}(a)$	$K_{\mathrm{Car}_7}(a)$	$K_{\mathrm{Car}_8}(a)$	$K_{\mathrm{Car}_9}(a)$	$K_{\mathrm{Car}_{10}}(a)$	OEIS
(0,1,. . . ,1)	1 $T(1,0)$	2 $T(2,0)$	5 $T(3,0)$	14 $T(4,0)$	42 $T(5,0)$	132 $T(6,0)$	429 $T(7,0)$	1430 $T(8,0)$	A000108
(1,1,. . . ,1)	1 $T(2,1)$	5 $T(3,1)$	21 $T(4,1)$	84 $T(5,1)$	330 $T(6,1)$	1287 $T(7,1)$	5005 $T(8,1)$	19448 $T(9,1)$	A002054
(2,1,. . . ,1)	1 $T(3,2)$	9 $T(4,2)$	56 $T(5,2)$	300 $T(6,2)$	1485 $T(7,2)$	7007 $T(8,2)$	32032 $T(9,2)$	143208 $T(10,2)$	A002055
(3,1,. . . ,1)	1 $T(4,3)$	14 $T(5,3)$	120 $T(6,3)$	825 $T(7,3)$	5005 $T(8,3)$	28028 $T(9,3)$	148512 $T(10,3)$	755820 $T(11,3)$	A002056
(4,1,. . . ,1)	1 $T(5,4)$	20 $T(6,4)$	225 $T(7,4)$	1925 $T(8,4)$	14014 $T(9,4)$	91728 $T(10,4)$	556920 $T(11,4)$	3197700 $T(12,4)$	A007160
(5,1,. . . ,1)	1 $T(6,5)$	27 $T(7,5)$	385 $T(8,5)$	4004 $T(9,5)$	34398 $T(10,5)$	259896 $T(11,5)$	1790712 $T(12,5)$	11511720 $T(13,5)$	A033280
(6,1,. . . ,1)	1 $T(7,6)$	35 $T(8,6)$	616 $T(9,6)$	7644 $T(10,6)$	76440 $T(11,6)$	659736 $T(12,6)$	5116320 $T(13,6)$	36581688 $T(14,6)$	A033281
(7,1,. . . ,1)	1 $T(8,7)$	44 $T(9,7)$	936 $T(10,7)$	13650 $T(11,7)$	157080 $T(12,7)$	1534896 $T(13,7)$	13302432 $T(14,7)$	105172353 $T(15,7)$	—
(8,1,. . . ,1)	1 $T(9,8)$	54 $T(10,8)$	1365 $T(11,8)$	23100 $T(12,8)$	302940 $T(13,8)$	3325608 $T(14,8)$	32008977 $T(15,8)$	278397405 $T(16,8)$	—
(9,1,. . . ,1)	1 $T(10,9)$	65 $T(11,9)$	1925 $T(12,9)$	37400 $T(13,9)$	554268 $T(14,9)$	6789783 $T(15,9)$	72177105 $T(16,9)$	687401000 $T(17,9)$	—
(10,1,. . . ,1)	1 $T(11,10)$	77 $T(12,10)$	2640 $T(13,10)$	58344 $T(14,10)$	969969 $T(15,10)$	13180167 $T(16,10)$	153977824 $T(17,10)$	1599111800 $T(18,10)$	—


1 T(1,0) 2 T(2,0) 5 T(3,0) 14 T(4,0) 42 T(5,0) 132 T(6,0) 429 T(7,0) 1430 T(8,0)
1 T(2,1) 5 T(3,1) 21 T(4,1) 84 T(5,1) 330 T(6,1) 1287 T(7,1) 5005 T(8,1) 19448 T(9,1)
1 T(3,2) 9 T(4,2) 56 T(5,2) 300 T(6,2) 1485 T(7,2) 7007 T(8,2) 32032 T(9,2) 143208 T(10,2)
1 T(4,3) 14 T(5,3) 120 T(6,3) 825 T(7,3) 5005 T(8,3) 28028 T(9,3) 148512 T(10,3) 755820 T(11,3)
1 T(5,4) 20 T(6,4) 225 T(7,4) 1925 T(8,4) 14014 T(9,4) 91728 T(10,4) 556920 T(11,4) 3197700 T(12,4)
1 T(6,5) 27 T(7,5) 385 T(8,5) 4004 T(9,5) 34398 T(10,5) 259896 T(11,5) 1790712 T(12,5) 11511720 T(13,5)
1 T(7,6) 35 T(8,6) 616 T(9,6) 7644 T(10,6) 76440 T(11,6) 659736 T(12,6) 5116320 T(13,6) 36581688 T(14,6)
1 T(8,7) 44 T(9,7) 936 T(10,7) 13650 T(11,7) 157080 T(12,7) 1534896 T(13,7) 13302432 T(14,7) 105172353 T(15,7)
1 T(9,8) 54 T(10,8) 1365 T(11,8) 23100 T(12,8) 302940 T(13,8) 3325608 T(14,8) 32008977 T(15,8) 278397405 T(16,8)
1 T(10,9) 65 T(11,9) 1925 T(12,9) 37400 T(13,9) 554268 T(14,9) 6789783 T(15,9) 72177105 T(16,9) 687401000 T(17,9)
1 T(11,10) 77 T(12,10) 2640 T(13,10) 58344 T(14,10) 969969 T(15,10) 13180167 T(16,10) 153977824 T(17,10) 1599111800 T(18,10)

$\downarrow$

$$\begin{tikzcd}
1 \arrow[r] & 2 \arrow[r] & 1 \arrow[r] & 5 \arrow[r] & 5 \arrow[r] & 1 \arrow[r] & 14 \arrow[r] & 21 \arrow[r] & 9 \arrow[r] & 1 \arrow[r] & 42 \\
{} \arrow[r] & 84 \arrow[r] & 56 \arrow[r] & 14 \arrow[r] & 1 \arrow[r] & 132 \arrow[r] & 330 \arrow[r] & 300 \arrow[r] & 120 \arrow[r] & 20 \arrow[r] & 1 \arrow[r] & 429 \\
{} \arrow[r] & 1287 \arrow[r] & 1485 \arrow[r] & 825 \arrow[r] & 225 \arrow[r] & 27 \arrow[r] & 1 \arrow[r] & 1430 \arrow[r] & 5005 \arrow[r] & 7007 \arrow[r] & 5005 \arrow[r] & 1925 \\
{} \arrow[r] & 385 \arrow[r] & 35 \arrow[r] & 1 \arrow[r] & 4862 \arrow[r] & 19448 \arrow[r] & 32032 \arrow[r] & 28028 \arrow[r] & 14014 \arrow[r] & 4004 \arrow[r] & 616 \arrow[r] & 44 \\
{} \arrow[r] & 1 \arrow[r] & 16796 \arrow[r] & 75582 \arrow[r] & 143208 \arrow[r] & 148512 \arrow[r] & 91728 \arrow[r] & 34398 \arrow[r] & 7644 \arrow[r] & 936 \arrow[r] & 54 \arrow[r] & 1 \\
{} \arrow[r] & 58786 \arrow[r] & 293930 \arrow[r] & 629850 \arrow[r] & 755820 \arrow[r] & 556920 \arrow[r] & 259896 \arrow[r] & 76440 \arrow[r] & 13650 \arrow[r] & 1365 \arrow[r] & 65 \arrow[r] & 1 \\
{} \arrow[r] & 208012 \arrow[r] & 1144066 \arrow[r] & 2735810 \arrow[r] & 3730650 \arrow[r] & 3197700 \arrow[r] & 1790712 \arrow[r] & 659736 \arrow[r] & 157080 \arrow[r] & 23100 \arrow[r] & 1925 \arrow[r] & 77 \\
{} \arrow[r] & 742900 \arrow[r] & 4457400 \arrow[r] & 11767536 \arrow[r] & 17978180 \arrow[r] & 17587350 \arrow[r] & 11511720 \arrow[r] & 5116320 \arrow[r] & 1534896 \arrow[r] & 302940 \arrow[r] & 37400 \arrow[r] & 2640 \\
{} \arrow[r] & 2674440 \arrow[r] & 17383860 \arrow[r] & 50220040 \arrow[r] & 84987760 \arrow[r] & 93486536 \arrow[r] & 70114902 \arrow[r] & 36581688 \arrow[r] & 13302432 \arrow[r] & 3325608 \arrow[r] & 554268 \arrow[r] & \cdots
\end{tikzcd}$$

Figure 11. Illustrations of the sequence A126216.

- When $a_2 = 2$, we have the following.

Table 5. Enumeration of $K_{\mathrm{Car}_n}(a)$ for vectors of the form $(a_1, 2, \ldots, 2)$.

Vector (a)	$K_{\mathrm{Car}_3}(a)$	$K_{\mathrm{Car}_4}(a)$	$K_{\mathrm{Car}_5}(a)$	$K_{\mathrm{Car}_6}(a)$	$K_{\mathrm{Car}_7}(a)$	$K_{\mathrm{Car}_8}(a)$	$K_{\mathrm{Car}_9}(a)$	$K_{\mathrm{Car}_{10}}(a)$	OEIS
(0,2,...,2)	1	3	12	55	273	1428	7752	43263	A001764
(1,2,...,2)	1	7	45	286	1820	11628	74613	480700	A236194
(2,2,...,2)	1	12	110	910	7140	54264	403788	2960100	—
(3,2,...,2)	1	18	220	2275	21420	189924	1615152	13320450	—
(4,2,...,2)	1	25	390	4900	54264	553014	5313000	48841650	—
(5,2,...,2)	1	33	637	9520	122094	1413258	15195180	154517220	—
(6,2,...,2)	1	42	980	17136	251370	3272808	39073320	436679100	—
(7,2,...,2)	1	52	1440	29070	482790	7013160	92355120	1128087675	—
(8,2,...,2)	1	63	2040	47025	876645	14109810	203783580	2707410420	—
(9,2,...,2)	1	75	2805	73150	1519518	26936910	424549125	6109028640	—
(10,2,...,2)	1	88	3762	110110	2532530	49189140	842305464	13077846496	—

We have also considered the case of $a_2 = 2$, and we obtained many new sequences. However, only a very small number of these sequences can find existing sequences and known combinatorial objects on the OEIS, making it difficult to establish new combinatorial bijections.

- When $a_2 = 3$, we have the following.

Table 6. Enumeration of $K_{\mathrm{Car}_n}(a)$ for vectors of the form $(a_1, 3, \ldots, 3)$.

Vector (a)	$K_{\mathrm{Car}_3}(a)$	$K_{\mathrm{Car}_4}(a)$	$K_{\mathrm{Car}_5}(a)$	$K_{\mathrm{Car}_6}(a)$	$K_{\mathrm{Car}_7}(a)$	$K_{\mathrm{Car}_8}(a)$	$K_{\mathrm{Car}_9}(a)$	$K_{\mathrm{Car}_{10}}(a)$	OEIS
(0,3,...,3)	1	4	22	140	969	7084	53820	420732	A002293
(1,3,...,3)	1	9	78	680	5985	53130	475020	4272048	—
(2,3,...,3)	1	15	182	2040	21945	230230	2375100	24208272	—
(3,3,...,3)	1	22	350	4845	61985	753480	8835372	100867800	—
(4,3,...,3)	1	30	600	9975	148764	2063880	27185760	344341800	—
(5,3,...,3)	1	39	952	18620	318780	4987710	73099488	1019251728	—
(6,3,...,3)	1	49	1428	32340	627900	10972962	177527328	2707044912	—
(7,3,...,3)	1	60	2052	53130	1158300	22428252	397906080	6598421973	—
(8,3,...,3)	1	72	2850	83490	2027025	43195152	835602768	14996413575	—
(9,3,...,3)	1	85	3850	126500	3396393	79191112	1662220560	32149174200	—
(10,3,...,3)	1	99	5082	185900	5486481	139267128	3158219064	65584315368	—

We have also considered the case of $a_2 = 3$, and we obtained many new sequences. But we did not find any existing combinatorial sequences among the new sequences, so no new combinatorial bijections were established. Perhaps when a_2 takes other values, we can obtain valuable sequences, but this will also require a lot of work. We will think deeply about this problem in the future.

Second, the bijection between pseudo-ladder diagrams and Dyck paths provides a purely combinatorial perspective on the Lidskii formula. While Baldoni and Vergne originally derived it through residue computations, and Mészáros and Morales analyzed $\mathcal{F}_G(a)$ via polytope subdivisions, our approach achieves the same result entirely combinatorially, highlighting the inherent structure of flow polytopes and offering fresh insights into their enumerative properties.

It is also worth noting that the discrete geometric structures underlying flow polytopes may have latent relevance to optimization and data flow in learning-based image deblurring models; see, for instance, MC-Blur [30], DBLRNet [31], and the survey *Deep Image Deblurring: A Survey* [32]. Although exploring this connection lies beyond the scope of the present paper, it highlights potential interdisciplinary links and suggests promising directions for future research.

Use of AI tools declaration

The authors declare they have not used artificial intelligence (AI) tools in the creation of this article.

Acknowledgments

The author would like to thank all those who provided valuable suggestions in the early stages of this research. Special thanks are also extended to the anonymous reviewers for their constructive comments, which greatly helped improve the quality of this paper.

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency, Vol. A. Paths, Flows, Matchings. Chapters 1–38*, Algorithms Combin., 24A, Springer-Verlag, Berlin, (2003), Chapter 13, 198–216.
2. J. E. Humphreys, *Introduction to Lie Algebras and Representation Theory*, Second printing, revised, Graduate Texts in Mathematics, 9, Springer-Verlag, New York–Berlin, (1978), xii+171 pp.
3. K. Mészáros, A. H. Morales, B. Rhoades, The polytope of Tesler matrices, *Selecta Math. (N.S.)*, **23** (2017), 425–454. https://doi.org/10.1007/s00029-016-0241-2
4. L. Hille, Quivers, cones and polytopes, *Linear Algebra Appl.*, **365** (2003), 215–237. https://doi.org/10.1016/S0024-3795(02)00406-8
5. W. Baldoni, M. Vergne, Kostant partition functions and flow polytopes, *Transform. Groups*, **13** (2008), 447–469. https://doi.org/10.1007/s00031-008-9019-8
6. R. P. Stanley, A. Postnikov, Acyclic flow polytopes and Kostant's partition function, in *Conference Transparencies*, 2000.
7. M. Beck, S. Robins, *Computing the Continuous Discretely: Integer-Point Enumeration in Polyhedra*, Undergrad. Texts Math., Springer, New York, (2007), xviii+226 pp.
8. R. Simion, Convex polytopes and enumeration, *Adv. Appl. Math.*, **18** (1997), 149–180. https://doi.org/10.1006/aama.1996.0505
9. B. V. Lidskii, The Kostant function of the system of roots A_n, *Funktsional. Anal. i Prilozhen.*, **18** (1984), 76–77. https://doi.org/10.1007/BF01076370
10. A. Postnikov, *Flows in Networks*, PhD thesis, Massachusetts Institute of Technology, 1997.
11. D. Zeilberger, Proof of the alternating sign matrix conjecture, *Electron. J. Combin.*, **3** (1996), 84. https://doi.org/10.37236/1271
12. K. Mészáros, A. H. Morales, Volumes and Ehrhart polynomials of flow polytopes, *Math. Z.*, **293** (2019), 1369–1401. https://doi.org/10.1007/s00209-019-02283-z
13. J. B. Liu, L. Guan, J. Cao, L. Chen, Coherence analysis for a class of polygon networks with the noise disturbance, *IEEE Trans. Syst. Man Cybern. Syst.*, **55** (2025), 4718–4727. https://doi.org/10.1109/TSMC.2025.3559326
14. C. Benedetti, R. S. González D'León, C. R. H. Hanusa, P. E. Harris, A. Khare, A. H. Morales, et al., A combinatorial model for computing volumes of flow polytopes, *Trans. Amer. Math. Soc.*, **372** (2019), 3369–3404. https://doi.org/10.1090/tran/7743
15. K. Mészáros, A. H. Morales, Flow polytopes of signed graphs and the Kostant partition function, *Int. Math. Res. Not.*, **3** (2015), 830–871. https://doi.org/10.1093/imrn/rnt212
16. J. Jang, J. S. Kim, Volumes of flow polytopes related to caracol graphs, *Electron. J. Combin.*, **27** (2020), 21. https://doi.org/10.37236/9187
17. I. Fischer, Counting integer points in polytopes associated with directed graphs, *Adv. Appl. Math.*, **79** (2016), 125–153. https://doi.org/10.1016/j.aam.2016.04.008

18. K. Mészáros, A. H. Morales, J. Striker, On flow polytopes, order polytopes, and certain faces of the alternating sign matrix polytope, *Discrete Comput. Geom.*, **62** (2019), 128–163. https://doi.org/10.1007/s00454-019-00073-2

19. C. S. Chan, D. P. Robbins, D. S. Yuen, On the volume of a certain polytope, *Experiment. Math.*, **9** (2000), 91–99. http://projecteuclid.org/euclid.em/1046889594

20. R. P. Stanley, A survey of alternating permutations, in *Combinatorics and Graphs*, (2010), 165–196. https://doi.org/10.1090/conm/531/10466

21. R. P. Stanley, J. Pitman, A polytope related to empirical distributions, plane trees, parking functions, and the associahedron, *Discrete Comput. Geom.*, **27** (2002), 603–634. https://doi.org/10.1007/s00454-002-2776-6

22. N. J. A. Sloane, *On-line Encyclopedia of Integer Sequences (OEIS)*, 2018. Available from: https://oeis.org.

23. E. Barcucci, A. Bernini, R. Pinzani, Exhaustive generation of some lattice paths and their prefixes, *Theoret. Comput. Sci.*, **878/879** (2021), 47–52. https://doi.org/10.1016/j.tcs.2020.12.013

24. R. R. X. Du, Y. Nie, X. Sun, Enumerations of humps and peaks in (k,a)-paths and (n,m)-Dyck paths via bijective proofs, *Discrete Appl. Math.*, **190/191** (2015), 42–49. https://doi.org/10.1016/j.dam.2015.04.005

25. R. Cori, A. Frosini, G. Palma, E. Pergola, S. Rinaldi, On doubly symmetric Dyck words, *Theor. Comput. Sci.*, **896** (2021), 79–97. https://doi.org/10.1016/j.tcs.2021.10.006

26. T. Mansour, E. Y. P. Deng, R. R. X. Du, Dyck paths and restricted permutations. *Discrete Appl. Math.*, **154** (2006), 1593–1605. https://doi.org/10.1016/j.dam.2006.02.004

27. R. A. Sulanke, Three dimensional Narayana and Schröder numbers, *Theor. Comput. Sci.*, **346** (2005), 455–468. https://doi.org/10.1016/j.tcs.2005.08.014

28. L. Yang, S. L. Yang, A Chung-Feller property for the generalized Schröder paths, *Discrete Math.*, **343** (2020), 111826. https://doi.org/10.1016/j.disc.2020.111826

29. L. Yang, Y. Y. Zhang, S. L. Yang, The halves of Delannoy matrix and Chung-Feller properties of the m-Schröder paths, *Linear Algebra Appl.*, **685** (2024), 138–161. https://doi.org/10.1016/j.laa.2023.12.021

30. K. H. Zhang, T. Wang, W. H. Luo, W. Q. Ren, B. Stenger, W. Liu, et al., Mc-blur: A comprehensive benchmark for image deblurring, *IEEE Trans. Circuits Syst. Video Technol.*, **34** (2024), 3755–3767. https://doi.org/10.1109/TCSVT.2023.3319330

31. K. H. Zhang, W. H. Luo, Y. R. Zhong, L. Ma, W. Liu, H. D. Li, Adversarial spatio-temporal learning for video deblurring, *IEEE Trans. Image Process.*, **28** (2018), 291–301. https://doi.org/10.1109/TIP.2018.2867733

32. K. H. Zhang, W. Q. Ren, W. H. Luo, W. S. Lai, B. Stenger, M. H. Yang, et al., Deep image deblurring: A survey, *Int. J. Comput. Vis.*, **130** (2022), 2103–2130. https://doi.org/10.1007/s11263-022-01633-5

Appendix 1: Python code for the number of vector partitions

```python

import copy
'''
@description Based on the number of nodes entered,
             a list of enumeration results of generated edges is
             returned
             The list consists of 3n-1 edges, each of which is a
             separate array
Example:     n = 5
             then [1,1,1,0,0]represent the edge a1 + a2 + a3
@params a1 The first number
@params a2 The second number, which defaults to 1
@params an The last number, which identifies the number of parameters
           an = -(a1 + n * a2) < 0,
           the number of parameters is n + 1
@return
'''
def enumEdge(a1, a2, an):
    lastParam = -an
    n = (lastParam - a1) // a2 + 1
    edgeList = []
    edgeSet = set()
    pld = []
    for i in range(n):
        eachRow = [-1] * lastParam
        for j in range(lastParam):
            if j >= i:
                eachRow[j] = 0
        pld.append(eachRow)
    nowEdge = pldTransToStr(pld)
    if nowEdge not in edgeSet:
        edgeSet.add(nowEdge)
        edgeList.append(nowEdge)
        print(nowEdge)
    count = 0
    while True:
        nowRow = 0
        nowCol = 0
        while nowRow < n and nowCol < lastParam:
            if nowCol > nowRow and not nowRow == n - 1:
                if pld[nowRow][nowCol] == 1:
                    pld[nowRow][nowCol] = 0
                elif pld[nowRow][nowCol] == 0:
                    pld[nowRow][nowCol] = 1
                    break
            if nowCol < lastParam - 1:
                nowCol += 1
            elif nowCol == lastParam - 1:
                nowCol = 0
                nowRow += 1
        if nowRow == n:
            break
        # print(pld)
        nowEdge = pldTransToStr(pld)
        if nowEdge == ””:
            continue
        if nowEdge not in edgeSet:
            count += 1
            edgeSet.add(nowEdge)
            edgeList.append(nowEdge)
            print(count,”:”,nowEdge,”\n”)
    return edgeList

’’’
@description Converts the two-dimensional array represented by pld
             into a mathematical formula of the corresponding string
             type
@params
@return
’’’
def pldTransToStr(pld):
    resultStr = ”” # record  the last generated formula type:string
    row = len(pld)
    col = len(pld[0])
    singleNum = []
    zeroNum = 0
    # obtain vector partitions that is not combined
    for i in range(row):
        for j in range(col):
            if pld[i][j] == 0:
                if i == 0:
                    zeroNum += 1
                else:
                    if pld[i - 1][j] == 0:
                        zeroNum += 1
        singleNum.append(zeroNum)
        zeroNum = 0
    for i in range(row):
     if singleNum[row - i - 1] == 1:
       resultStr += "a" + str(i + 1) + "+"
        elif singleNum[row - i - 1] == 0:
         continue
     else:
       resultStr += str(singleNum[row - i - 1]) + "*a" + str(i + 1) + "+"

    # the case of obtaining vector partitions combinations
    newPld = copy.deepcopy(pld)
    combDict = {}
    nowList = []
    nowKey = ""
    for i in range(row):
        for j in range(col):
            if newPld[i][j] == 1:
                # print(i,"aaa",j)
                nowList.append(row - i)
                newPld[i][j] = 0
                nowIndex = i + 1
                while newPld[nowIndex][j] == 1:
                    nowList.append(row - nowIndex)
                    newPld[nowIndex][j] = 0
                    nowIndex += 1
                nowList.append(row - nowIndex)
                nowKey += str(nowList[len(nowList) - 1]) + str(nowList[0])
                if not nowList[len(nowList) - 1] == 1:
                    if not nowList[0] == row:
                        return ""
                elif not nowList[0] == row:
                    if not nowList[len(nowList) - 1] == 1:
                        return ""
                elif nowList[len(nowList) - 1] == 1 and nowList[0] == row:
                    return ""
                combDict[nowKey] = combDict.get(nowKey, 0) + 1
                nowKey = ""
                nowList = []
    for i in range(row):
        for j in range(i + 1, row):
            nowKey = str(i + 1) + str(j + 1)
            # print(nowKey, ":", combDict.get(nowKey, 0))
            if combDict.get(nowKey, 0) == 0:
                continue
            if combDict.get(nowKey) == 1:
                resultStr += "("
            else:
                resultStr += str(combDict.get(nowKey)) + "("
            for k in range(i + 1, j + 1):
                resultStr += "a" + str(k) + "+"
            resultStr += "a" + str(j + 1) + ")" + "+"
    return resultStr[:-1]




if __name__ == '__main__':
    '''
    Describe the pld diagram, -1 is used for placeholders,
    indicating that there are no points at that point
    0 indicates that there is no vertical step at present position
    1 indicates that there is a vertical step at present position
    '''

    # pld = [[0, 0, 1, 1, 1, 1, 1],
    #         [-1, 0, 0, 1, 1, 1, 1],
    #         [-1, -1, 0, 0, 0, 1, 1],
    #         [-1, -1, -1, 0, 0, 0, 1],
    #         [-1, -1, -1, -1, 0, 0, 0]]
    # 2*a5
    # print(pldTransToStr(pld))

    enumEdge(3, 1, -7)
    # print(enumList)
    # for [x, y] in enumList:
    #     if x == y:
    #         print("a"+str(x))
    #     else:
    #         p = ""
    #         for i in range(x, y):
    #             p += "a"
    #             p += str(i)
    #             p += "+"
    #           p += "a" + str(y)
    #           print(p)


    # pld = pldEnum(3, 1, -7)
    # print("Generated Pld diagram", pld)
```

Appendix 2: Python code for two bijections

```
'''
An instance like this:
Input:
c = [3, 1, 1, 1, 1]
3a1 + 4a2 + 5a3 + 6a4 + 7a5 =
2a1+a2+2a5+(a4+a5)+2(a3+a4+a5)+2(a2+a3+a4+a5)+(a1+a2+a3+a4)

Output:
pld = [
    [-1, -1, -1, -1, 3, 3, 1],
    [-1, -1, -1,  3, 1, 1, 1],
    [-1, -1,  1,  1, 1, 1, 1],
    [-1,  1,  1,  1, 1, 1, 1],
    [ 3,  2,  1,  1, 1, 1, 0]
]
-1 indicates that there is no line segment here;
3 indicates that there have solid gray  points;
1 indicates that there have line segment;
2 indicates that there are both line segments and solid gray points here.
'''
def tranStrToDict(partition):
    partDict = {}
    n = len(partition)
    i = 0
    nowPower = 1
    while i < n:
    if partition[i] == '(':
      nowIndex = i
        while nowIndex >= 0 and partition[nowIndex] != '+':
          nowIndex -= 1
            if nowIndex + 1 != i:
                if nowIndex == 0:
                    nowPower = int(partition[nowIndex: i])
                else:
                    nowPower = int(partition[nowIndex + 1: i])
            else:
                nowPower = 1
            while i < n and partition[i] != 'a':
                i += 1
            startIndex = i + 1
            i += 1
            while i < n and partition[i] != '+':
                i += 1
            nowMin = int(partition[startIndex: i])
            while partition[i] != ')':
                i += 1
            endIndex = i
            while partition[endIndex] != 'a':
                endIndex -= 1
            nowMax = int(partition[endIndex + 1: i])
            nowKey = str(nowMin) + "_" + str(nowMax)
            partDict[nowKey] = nowPower
        i += 1

    return partDict

def partitionToPld(c ,partition):
 partDict = tranStrToDict(partition)
 c1 = c[0]
 c2 = c[1]
 row = len(c) # number of rows of the PLD diagram
 col = c1 + c2 * (row - 1) # number of columns of the PLD diagram

    # init pld
    pld = []
    for i in range(row):
        ci = c1 + i * c2
        nowRow = [0] * col
        for j in range(col - ci):
            nowRow[j] = -1
        pld.append(nowRow)

    # the second type edge
    nowMax = row
    nowCol = 1
    nowMin = nowMax - 1
```

```python
    while nowMin > 1:
        nowKey = str(nowMin) + "_" + str(nowMax)
        n = partDict.get(nowKey, 0)
        if n != 0:
            nowCol = max(nowCol, row - nowMin)
            while n > 0:
                n -= 1
                for i in range(nowMin - 1, nowMax):
                    pld[i][nowCol] = 1
                nowCol += 1
        nowMin -= 1

    # the third type edge
    nowMin = 1
    nowMax = nowMin + 1
    while nowMax < row:
        nowKey = str(nowMin) + "_" + str(nowMax)
        n = partDict.get(nowKey, 0)
        nowCol = max(nowCol, col - row + 1)
        if n != 0:
            while n > 0:
                n -= 1
                for i in range(nowMin - 1, nowMax):
                    pld[i][nowCol] = 1
                nowCol += 1
        nowMax += 1

    # the first type edge, gray point
    for i in range(row):
        count = 0
        ci = c1 + c2 * i
        for j in range(col):
            if pld[i][j] == 1:
                count += 1
        count = ci - count
        for j in range(col):
            if pld[i][j] == 1 and count != 0:
                pld[i][j] = 2
                count -= 1
            elif pld[i][j] == 0 and count != 0:
                pld[i][j] = 3
                count -= 1
    return pld
```

```

def pldToDyckPath(pld, m, n):
    pld_x = len(pld[0])
    now_x = 0
    row = len(pld)
    now_y = len(pld) - 1
    dyck_tran_path = [(0,0)]
    while now_x < pld_x and now_x < m:
    flag = False
    if pld[now_y][now_x] == 1 or pld[now_y][now_x] == 2:
     if now_y != 0:
      if pld[now_y - 1][now_x] == 1 or pld[now_y - 1][now_x] == 2:
                    now_y -= 1
                    flag = True
        if not flag:
            now_x += 1
        dyck_tran_path.append((now_x,row - now_y - 1))
    now_y = row - now_y - 1
    if now_x < pld_x:
      # it means that now_x has reached the boundary
      and can go straight up
        while now_y < n:
            now_y += 1
            dyck_tran_path.append((now_x, now_y))
    else:
        # indicates that the given PLD diagram has been traversed
        while now_x < m:
            now_x += 1
            dyck_tran_path.append((now_x, now_y))
        while now_y < n:
            now_y += 1
            dyck_tran_path.append((now_x, now_y))
    dyck_path = []
    while len(dyck_tran_path) > 0:
        (x, y) = dyck_tran_path.pop()
        dyck_path.append((m - x, n - y))
    return dyck_path

def dyckPathToPld(dyckPath):
    # step 1: rotate 180 degrees
    dyck_path = []
    (m, n) = dyckPath[len(dyckPath) - 1]
    while len(dyckPath) > 0:
        (x, y) = dyckPath.pop()
        dyck_path.append((m - x, n - y))
    # step 2: delete all the east edges
    sec_path = []
    last_x = 0
    last_y = 0
    isLast = False
    for (x, y) in dyck_path:
        if x == 0 and y == 0:
            continue
        if y - last_y == 1:
            isLast = True
            sec_path.append((last_x, last_y))
        elif x - last_x == 1:
            if isLast:
                sec_path.append((last_x, last_y))
            isLast = False
        (last_x, last_y) = (x, y)
    # step 3: delete the last north step
    (last_x, last_y) = sec_path.pop()
    (x, y) = sec_path.pop()
    while last_y - y == 1 and last_x - x == 0:
        (last_x, last_y) = (x, y)
        (x, y) = sec_path.pop()
    sec_path.append((x, y))
    # step 4:correspond the remaining north step to the PLD diagram
    row = y + 1
    col = x + 1
    pld = []
    for i in range(row):
        ci = col - (row - i - 1)
        nowRow = [0] * col
        for j in range(col - ci):
            nowRow[j] = -1
        pld.append(nowRow)
    for (x, y) in sec_path:
        pld[row - y - 1][x] = 1
    # step 5: extend the last north step down
    for i in range(0, row - 1):
        pld[i][col - 1] = 1
    # step 6: extend the other north steps down
    for j in range(1, col - 1):
        i = 1
        while i < row and pld[i][j] != 1:
            i += 1
        if i != row:
            while i < row:
                pld[i][j] = 1
                i += 1
        else:
            nowRow = row - 1
            while nowRow > 0:
                if pld[nowRow][j - 1] == -1:
                    break
                pld[nowRow][j] = pld[nowRow][j - 1]
                nowRow -= 1
    # step 7:add remaining points
    for i in range(row):
        count = 0
        ci = col - (row - i - 1)
        for j in range(col):
            if pld[i][j] == 1:
                count += 1
        count = ci - count
        for j in range(col):
            if pld[i][j] == 1 and count != 0:
                pld[i][j] = 2
                count -= 1
            elif pld[i][j] == 0 and count != 0:
                pld[i][j] = 3
                count -= 1
    return pld

def pldToSchroder(pld):
    pld_x = len(pld[0])
    now_x = 0
    now_y = len(pld) - 1
    schroder_tran_path = []
    east_step = 0
    add_x = 0
    add_y = 0
    while now_x < pld_x and now_y >= 0:
        schroder_tran_path.append((add_x,add_y))
        flag = False
        if pld[now_y][now_x] == 1 or pld[now_y][now_x] == 2:
            if now_y != 0:
                if pld[now_y - 1][now_x] == 1
                or pld[now_y - 1][now_x] == 2:
                    now_y -= 1
                    add_y += 1
                    flag = True
                    east_step = 0
        if not flag:
            now_x += 1
            add_x += 1
            east_step += 1
            if east_step == 2:
                east_step = 0
                add_y += 2
                add_x += 1

    schroder_tran_path.append((add_x,add_y))
    add_y += 1
    schroder_tran_path.append((add_x,add_y))

    ### rotate 180 degrees
    schroder_path = []
    (m, n) = schroder_tran_path[len(schroder_tran_path) - 1]
    while len(schroder_tran_path) > 0:
        (x, y) = schroder_tran_path.pop()
        schroder_path.append((m - x, n - y))
    return schroder_path

if __name__ == '__main__':
    c = [3, 1, 1, 1, 1]
    print("c: ", c)
    # partition = "2a1+2a2+a3+a4+3a5+(a4+a5)+2(a3+a4+a5)+(a2+a3+a4+a5)
    +(a1+a2+a3+a4)"
    partition = "2a1+a2+2a5+(a4+a5)+2(a3+a4+a5)+2(a2+a3+a4+a5)
    +(a1+a2+a3+a4)"
    print("partition:",partition)
    pld = partitionToPld(c, partition)
    print("pld:")
    for i in pld:
        print(i)
    dyckPath = pldToDyckPath(pld, 7, 12)
    print("Dyck Path: ", dyckPath)
    pld_tran = dyckPathToPld(dyckPath)
    print("Dyck Path To pld:")
    for i in pld_tran:
        print(i)
    schroder_path = pldToSchroder(pld)

    print("schroder_path: ", schroder_path)
```