ABSTRACT

SINGANALLUR VENKATARAMAN, SRINIVASAN. Enumerating $2 \times 2 \times n$ Solid Partitions and Counting Linear Extensions. (Under the direction of Professor Carla D. Savage).

The problem of enumerating integer partitions in three dimensions (solid partitions) is an unsolved problem in combinatorics. Generating functions are one of the most popular analytical tools used in enumerating partitions. Both ordinary partitions and plane partitions have a closed form for the generating function that enumerates them. However, there is no known closed form for the generating function of solid partitions, even for special cases. Our contribution, is the derivation of an explicit recurrence for the generating function of a special family of solid partitions, bounded by the $2 \times 2 \times n$ recursive structure.

Integer partitions can be represented as integer solutions to a set of linear inequalities. In this thesis, we use this representation for $2 \times 2 \times n$ solid partitions, and apply a set of rules called the "digraph methods", to derive a recurrence for the generating function. We implement this recurrence in Maple, do some optimizations and obtain the generating function for a few values of n. We also count the linear extensions of the $2 \times 2 \times n$ poset from this generating function, using the theory of P-partitions.

In recent years many methods have evolved for finding integer solutions to linear inequalities. For inequalities of the form, $s_a \geq s_b$ and $s_a > s_b$, which can be modeled as directed graphs, the "digraph methods" can be used. We are able to enumerate $2 \times 2 \times n$ solid partitions, which is a hard problem, using the digraph method, thereby demonstrating the power of this method. We also compare the digraph approach to a few other methods to enumerate $2 \times 2 \times n$ solid partitions.

Enumerating $2 \times 2 \times n$ Solid Partitions and Counting Linear Extensions

by
Srinivasan Singanallur Venkataraman

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Steffen Heber	Dr. Robert D. Rodman

Dr. Carla D. Savage
Chair of Advisory Committee

DEDICATION

To my parents, sister and close friends.

BIOGRAPHY

Srinivasan was born in Chennai, India on April 12th 1985. He earned his Bachelors degree in Computer Science and Engineering in 2006 from PSG College of Technology, Coimbatore, India. He is currently doing his masters in Computer Science at North Carolina State University, Raleigh, USA.

ACKNOWLEDGMENTS

I would like to thank my adviser, Dr. Carla Savage for her invaluable guidance in the making of this thesis. Her constant motivation, encouragement and valuable feedback have greatly helped me in the completion of this thesis. During the course of this work , she has imparted some priceless lessons, which I will always carry with me for the rest of my life. Her witty criticisms on my writing made our meetings livelier and brought out my mistakes in an interesting way. I would like to thank Dr. Robert Rodman and Dr. Steffen Heber for agreeing to be in my committee, for their support and the enthusiasm they showed in my work.

I would like to thank Dr. Mitch Harris for his interest in my work and his helpful correspondences over e-mail regarding counting linear extensions of the $2 \times 2 \times n$ poset. I would like to thank Dr. Frank Ruskey, University of Victoria, Canada, for teaching me about posets and giving some valuable suggestions during the early stages of my thesis. I would also like to thank Prashanth Iyer for working with me and giving many valuable ideas to improve my work. I would also like to thank my friends for proofreading my thesis. Finally, I am very grateful to my parents for their blessings and a special thanks to my sister who is always looking out for me, and to whom I credit many of the great things that have happened to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In this thesis, we study counting problems that are associated with the $2 \times 2 \times n$ digraph shown in Figure 1.1. Consider the *feasible labellings* of this digraph, that is, labellings of the vertices with non-negative integers such that, if there is an edge from y to z, the integer assigned to y is greater than or equal to the integer assigned to z. It is evident that there are infinitely many such feasible

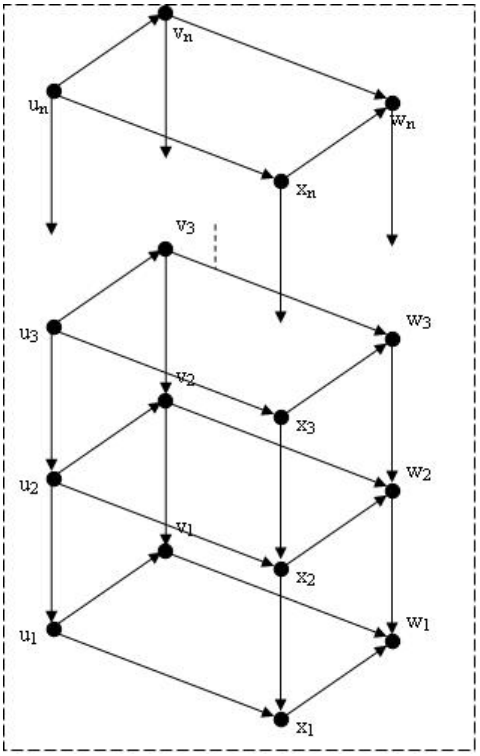

Figure 1.1: The $2 \times 2 \times n$ family

labellings. On the other hand, if these labellings are grouped according to the sum of the labels, then the number of feasible labellings that sum to a particular value, m, is finite. Each labelling that sums to m is called a $2 \times 2 \times n$ *solid partition* of m. To represent the number of such labellings we use a tool called *generating function*. The generating function, $G_{2\times 2\times n}(q)$, for $2 \times 2 \times n$ solid partitions, is a single variable power series in q, in which the coefficient of q^m is the number of $2 \times 2 \times n$ solid partitions of m.

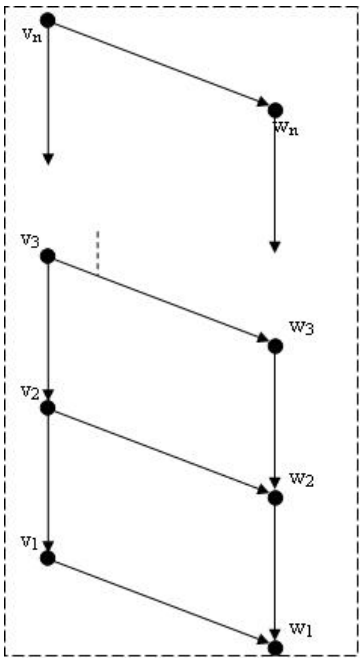

Figure 1.2: The $2 \times n$ family

In contrast, consider the $2 \times n$ digraph shown in Figure 1.2. The feasible labellings of this digraph that sum to m are called $2 \times n$ *plane partitions* of m. The generating function, $G_{2\times n}(q)$, of $2 \times n$ plane partitions is a power series where the coefficient of q^m is the number of $2 \times n$ plane partitions of m. Interestingly, $G_{2\times n}(q)$ has a closed form representation given by:

$$G_{2\times n}(q) = \frac{1}{(1-q)(1-q^2)^2(1-q^3)}.$$

The advantage of the closed form is that we can expand this rational function as a power series of q and look up the number of $2 \times n$ plane partitions of any integer $m \geq 0$ from the coefficient of q^m. Even $p \times n$ *plane partitions* have a closed form for the generating function, which will be shown in Section 3.1.1.

However, for the general case of $h \times p \times n$ *solid partitions*, a closed form for the generating function

is not known. It is not known even for special cases like $3 \times 2 \times n$ solid partitions or $3 \times 3 \times n$ solid partitions. In this thesis, we will derive an explicit recurrence for the generating function of one such special case: $2 \times 2 \times n$ solid partitions.

Going back to the digraph in Figure 1.1, consider the vertices of the digraph as a set and the edges as relations among the elements of the set. This set along with the relations is called the $2 \times 2 \times n$ *partially ordered set* (poset). A *linear extension* of this poset is an ordering of all the elements of the set such that y comes earlier than z, if there is an edge from y to z. The number of linear extensions, $\ell(P)$, of any poset P can be obtained from the generating function of the corresponding digraph, through the theory of P-partitions, which will be explained in Section 3.2.1.

The number of linear extensions, $\ell(P_{2\times n})$, of the $2 \times n$ poset, thus obtained from the generating function of $2 \times n$ plane partitions, has a nice formula given by:

$$\ell(P_{2\times n}) = \frac{(2n)!}{n!(n+1)!}.$$

A formula for the number of linear extensions, $\ell(P_{p\times n})$, of the $p \times n$ poset can also be got in the same way which will be shown in Section 3.2.2. But for solid partitions, neither the closed form representation for the generating function, nor a compact formula to count the linear extensions of the corresponding poset is known so far. The other aim in this thesis is to compute the number of linear extensions, $\ell(P_{2\times 2\times n})$, of the $2 \times 2 \times n$ poset from the generating function of $2 \times 2 \times n$ solid partitions. There have been past efforts in computing $\ell(P_{2\times 2\times n})$ and values up to $n = 6$ were computed and recorded in the Online Encyclopedia of Integer Sequences (OEIS) [1] at the time we began this work. These values are shown in Table 1.1.

To derive a recurrence for the generating function of $2 \times 2 \times n$ solid partitions, we will make use of a set of rules called the "digraph methods" [2] that will be discussed in Section 4.2. These digraph methods are used to find integer solutions to sets of linear inequalities of a particular type. We are able to represent $2 \times 2 \times n$ solid partitions as the set of solutions to such a system of linear

Table 1.1: Linear extensions of the $2 \times 2 \times n$ poset from OEIS

n	Linear extensions of the $2 \times 2 \times n$ poset
1	2
2	48
3	2452
4	183958
5	17454844
6	1941406508

inequalities.

Enumerating solid partitions is an example of an enumeration problem where the object can be defined by *diophantine equations*, that is, equations which allow only integer solutions. In recent years, there have been a number of methods that have been developed to find the generating function of integers defined by such diophantine equations. The digraph methods focus on inequalities that are of the form $s_a \geq s_b$ and $s_a > s_b$. There are a number of other methods to find integer solutions to such inequalities. We study a few such approaches and use them to enumerate $2 \times 2 \times n$ solid partitions. We compare these methods with our approach to understand their advantages and limitations.

1.1 Organization

In Chapter 2, we define some of the basic terminologies that will be used in the rest of this thesis. Chapter 3, defines the problem being solved and discusses the motivation behind solving it. In Chapter 4, we discuss the main technique that will be used to solve our problem. Chapter 5, illustrates our solution to derive a recurrence for the the generating function of $2 \times 2 \times n$ solid partitions using the digraph methods. In Chapter 6, we discuss the Maple implementation of the recurrence that we derived and some of the optimizations that we did. Chapter 7, discusses some other approaches to enumerate $2 \times 2 \times n$ solid partitions and presents a comparative study. Finally in Chapter 8 we record some interesting observations about $2 \times 2 \times n$ solid partitions and discuss possible future direction to this work.

Chapter 2

Background definitions

In this chapter, definitions of some of the basic terms that will be used in the rest of the thesis, are given.

2.1 Partition theory

2.1.1 Partitions and compositions

A *composition* of an integer m is a sequence of positive integers $s_1, s_2, s_3, ...$ where $s_1+s_2+s_3+... = m$. Here the order of the sequence is important. For example, the compositions of 4 are

$$(4), (1,3), (3,1), (2,2), (1,1,2), (1,2,1), (2,1,1), (1,1,1,1). \tag{2.1}$$

On the other hand, a *composition of an integer m into n non-negative parts* is defined as a sequence of non-negative integers $s_1, s_2, s_3, ..., s_n$ where $s_1 + s_2 + s_3 + ... + s_n = m$. Again, the order of the sequence is important. For example, there are 5 compositions of 4 into 2 non-negative parts:

$$(4,0), (0,4), (1,3), (3,1), (2,2). \tag{2.2}$$

A *partition* of an integer m is a sequence of positive integers $s_1, s_2, s_3, ...$ where $s_1+s_2+s_3+... = m$, but here the order of the sequence is not important, so we usually assume a non-increasing order i.e.

$s_1 \geq s_2 \geq s_3 \geq ....$ For example, there are 5 partitions of the integer 4

$$(4), (3,1), (2,2), (2,1,1), (1,1,1,1). \tag{2.3}$$

A *partition of an integer m into n non-negative parts* is defined as a sequence of non-negative integers $s_1, s_2, ..., s_n$ where $s_1 + s_2 + ... + s_n = m$ and $s_1 \geq s_2 \geq ... \geq s_n$. For example, there are 3 partitions of the integer 4 into 2 non-negative parts:

$$(4,0), (3,1), (2,2). \tag{2.4}$$

2.2 Generating functions

2.2.1 Generating functions

A generating function is a power series whose coefficients encode a sequence. Suppose we have a sequence of integers $s_0, s_1, s_2, s_3, ...$, then the generating function of this sequence, $G(q)$, is defined as

$$G(q) = s_0 + s_1 q + s_2 q^2 + s_3 q^3 ... = \sum_{n=0}^{\infty} s_n q^n.$$

This representation of the sequence can be very helpful when a closed form representation of $G(q)$ is known. For example, for the sequence $1, 1, 1, ...$, the generating function is

$$1 + q + q^2 + q^3 + ... = \frac{1}{(1-q)}.$$

A closed form can sometimes be derived from the recurrence that defines the sequence, if such a recurrence is known. Some of the advantages of having the generating function are:

- It gives a compact representation, provided by the closed form (if there is one).
- It may lead to a direct formula for the elements of the sequence.
- It can be used to prove two families of sets have the same size by proving the generating functions are equal.
- It can be combined in several ways with other generating functions to provide solutions to various problems associated with sequences.

- It can provide greater insight into an enumeration problem.
- It can sometimes be used to get the asymptotic estimates via complex analysis.

2.2.2 The generating function of partitions and compositions

The generating function of compositions into n positive parts is given by

$$\frac{q^n}{(1-q)^n}. \tag{2.5}$$

The coefficient of q^m gives the number of compositions of m into n positive parts. The generating function of compositions into n non-negative parts is given by

$$\frac{1}{(1-q)^n}. \tag{2.6}$$

The coefficient of q^m gives the number of compositions of m into n non-negative parts. For example, the number of compositions of 4 into 2 non-negative parts is given by the coefficient of q^4 in the expansion of

$$\frac{1}{(1-q)^2} = 1 + 2q + 3q^2 + 4q^3 + 5q^4 + ...$$

which is 5 as shown in Example 2.2.

The generating function of partitions into at most n parts is given by

$$\frac{1}{(1-q)(1-q^2)(1-q^3)...(1-q^n)}. \tag{2.7}$$

The coefficient of q^m gives the number of partitions of m into at most n parts. This is also the generating function of partitions into n non-negative parts. For example, the number of partitions of 4 into 2 non-negative parts is given by the coefficient of q^4 in the expansion of

$$\frac{1}{(1-q)(1-q^2)} = 1 + q + 2q^2 + 2q^3 + 3q^4 + ...$$

which is 3 as shown in Example 2.4.

2.3 Sequences defined by inequalities

Let C be a set of constraints of the form

$$C : c_{i,0} + c_{i,1}s_1 + c_{i,2}s_2 + c_{i,3}s_3 + ... + c_{i,n}s_n \geq 0, \;\; 1 \leq i \leq r$$

where the c_{ij}'s are integers. Let S_C be the set of non-negative integer solutions $s = (s_1, s_2, ..., s_n)$ to the given constraints C. The *weight* of s is defined as $s_1 + s_2 + ... + s_n$ and each of the s_i's are the *parts* of s. We are interested in obtaining the multivariate generating function of the set of sequences S_C given by

$$G(x_1, x_2, x_3, ..., x_n) = \sum_{s \in S_C} x_1^{s_1} x_2^{s_2} x_3^{s_3} ... x_n^{s_n}. \tag{2.8}$$

Instead of $G(x_1, x_2, ..., x_n)$, if we instead compute $G(qx_1, qx_2, ..., qx_n)$, then, in the resulting polynomial, the coefficient of q^m is a "list" of solutions of weight m and if we compute $G(q, q, q, ..., q)$, the coefficient of q^m in this polynomial is the number of solutions of weight m [3].

2.3.1 Partitions and compositions as sequences defined by inequalities

For the set of integer sequences S_C to represent compositions, the set of constraints C would be

$$s_1 \geq 0$$

$$s_2 \geq 0$$

$$\vdots$$

$$s_n \geq 0.$$

The integer solutions of weight m are the compositions of m into n non-negative parts. The multivariate generating function would be given by

$$\sum_{s_1 \geq 0, ..., s_n \geq 0} x_1^{s_1} ... x_n^{s_n} = (\sum_{s_1=0}^{\infty} x_1^{s_1}) ... (\sum_{s_n=0}^{\infty} x_n^{s_n}) = \frac{1}{1 - x_1} ... \frac{1}{1 - x_n}.$$

On substituting all x_i's with q, we get the univariate or "counting" generating function of compositions given by Equation 2.6.

For S_C to represent partitions, C would be

$$s_1 \geq s_2$$

$$s_2 \geq s_3$$

$$\vdots$$

$$s_{n-1} \geq s_n$$

$$s_n \geq 0.$$

All integer solutions that sum to m are the partitions of m into n non-negative parts. The multivariate generating function would be given by

$$\sum_{s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_n \geq 0} x_1^{s_1} \ldots x_n^{s_n} =$$

$$\sum_{s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_n \geq 0} x_1^{s_1 - s_2} (x_1 x_2)^{s_2 - s_3} (x_1 x_2 x_3)^{s_3 - s_4} \ldots (x_1 x_2 x_3 \ldots x_{n-1})^{s_{n-1} - s_n} (x_1 x_2 x_3 \ldots x_n)^{s_n}$$

$$= \left(\sum_{s_1 - s_2 = 0}^{\infty} x_1^{s_1 - s_2} \right) \left(\sum_{s_2 - s_3 = 0}^{\infty} (x_1 x_2)^{s_2 - s_3} \right) \ldots \left(\sum_{s_n = 0}^{\infty} (x_1 x_2 \ldots x_n)^{s_n} \right)$$

$$= \frac{1}{(1 - x_1)(1 - x_1 x_2) \ldots (1 - x_1 x_2 x_3 \ldots x_n)}.$$

Again on substituting all x_i's with q, we get the univariate generating function of partitions given by Equation 2.7.

2.4 Poset theory

2.4.1 Partially ordered sets

A partially ordered set P (poset) as defined in [4] is a set X and a binary relation $\leq_P$ on X that satisfies the following axioms:

1. For all $x \in X$, $x \leq_P x$ (*reflexivity*)
2. For $x, y \in X$, if $x \leq_P y$ and $y \leq_P x$, then $x = y$ (*antisymmetry*)
3. For $x, y, z \in X$, if $x \leq_P y$ and $y \leq_P z$, then $x \leq_P z$ (*transitivity*).

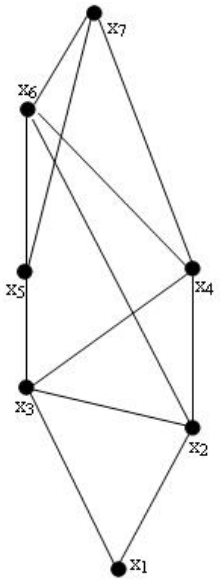

Figure 2.1: Hasse diagram for a poset

Two elements $x, y \in X$ are *comparable* in P if either $x \leq_P y$ or $y \leq_P x$, otherwise they are *incomparable* . For elements $x, y \in X$, we say that x *covers* y iff $y <_P x$ and there is no $z \in X$ such that $y <_P z <_P x$ (where $y <_P x$ implies that $y \leq_P x$ and $y \neq x$).

2.4.2 Hasse diagram

A Hasse diagram is a graphical representation of a poset P, where P is represented by the cover relationship between two elements x, y of the set X. A point for the elements and a line for the relationship is used according to these two rules:

1. If $x <_P y$ in the poset, then the point corresponding to x is drawn at a lower level than the point corresponding to y and
2. The line between any two points x and y is included iff x covers y or y covers x.

For example, Figure 2.1 shows a Hasse diagram of the poset $P(X, \leq_P)$ where

$$X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)$$

and the cover relations are specified by the line segments.

2.4.3 Linear extensions

A linear extension of a poset P is a permutation $w = x_1, x_2, ..., x_n$ of the elements of the set X, such that if $x_i \leq_P x_j$ in P, then $i \leq j$. In other words a linear extension is a total order on the set X that is consistent with the partial order. The set of linear extensions of P is represented by $L(P)$ and the number of linear extensions is represented by $\ell(P)$.

2.4.4 Order ideal

An order ideal I of a poset P is a subset of P such that if $x \in I$ and $y \leq_P x$, then $y \in I$. The set of all order ideals of any poset P along with the inclusion operator ($\subseteq$) is a poset itself. It is denoted by $J(P)$.

2.4.5 Chain

A chain L is a poset in which all the elements are comparable. A chain C in a poset P is a subset of P in which all the elements are comparable. C is *maximal* if there exists no $x \in P$ such that $C \cup \{x\}$ forms a chain. A chain C is *saturated* if there exists no element $z \in P - C$ such that $x \leq_P z \leq_P y$, for some $x, y \in C$ and $C \cup \{z\}$ forms a chain. In Figure 2.1, (x_1, x_3, x_5, x_7) is a chain whereas $(x_1, x_2, x_3, x_5, x_6, x_7)$ is a maximal saturated chain.

2.4.6 Representing posets as digraphs

A *directed graph* G (digraph) is an ordered pair (V, E) of vertices V and edges E such that every $e \in E$ has an initial and terminal vertex. The terminal vertex v_t of an edge e is denoted graphically by drawing an arrow toward v_t. A digraph is a *directed acyclic graph* (DAG) if for all vertices $v \in V$, there are no non-empty directed paths that start and end in v.

We can associate the Hasse diagram for a poset with a corresponding DAG by representing the set of elements X of the poset P by the set of vertices V and the line segments S denoting the cover relations by directed edges E. For a cover relation $x <_P y$ where $x, y \in X$, the corresponding edge $e \in E$ originates from the vertex that represents x and terminates at the vertex that represents y. The DAG shown in Figure 2.2 represents the Hasse diagram shown in Figure 2.1. In this thesis, we will represent a poset by its associated DAG. The number of linear extensions of the poset is equal to the number of topological sorts of the DAG, where a *topological sorting* of a directed acyclic graph

G is a linear ordering l of the vertices V of G, such that, for all $u, v \in V$ if there is an edge from u to v in G, then u comes earlier than v in l.

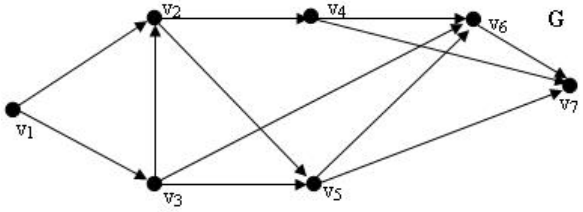

Figure 2.2: A directed acyclic graph G

2.5 Theory of P-partitions

In this section we talk about P-partitions which are a generalization of compositions (parts of an integer n where there is no order) and partitions (parts of an integer n where there is total order).

2.5.1 Poset labelling

A labelling of a poset $P = (X, \leq_P)$ is a bijection $\sigma : X \to [1...k]$ where k is the size of the set X. A labelling σ is *natural* if $\sigma(x_i) \leq \sigma(x_j)$ whenever $x_i \leq_P x_j$. A poset P along with a labelling σ is called a *labelled poset*. If σ is a natural labelling, then P is called a *naturally labelled poset*.

2.5.2 P-Partitions

If P is a labeled poset with a labelling σ, a (P, σ)-*partition* [5] of m, is an order-preserving[1] map $f : P \to [1, 2, 3...]$ satisfying the conditions:

1. If $x_i \leq_P x_j$ in P, then $f(x_i) \leq f(x_j)$.
2. If $x_i \leq_P x_j$ and $\sigma(x_i) > \sigma(x_j)$, then $f(x_i) < f(x_j)$.
3. $\sum_{x_i \in X} f(x_i) = m$.

If σ is a natural labelling, then f is just called *P-partition*. As in partitions and composition, the values of $f(x_i)$ are called the parts of m. Thus P-partitions are a generalization of partitions and compositions, where the former is obtained when P is a chain and the latter is obtained when P has no relations.

[1] Although Stanley originally defines them as order-reversing maps in [5]

2.5.3 Generating function of P-partitions

Let (P, σ) be a labelled poset $P = (X, \leq_P)$ with elements $X = \{X_1, X_2, ..., X_n\}$. We define the multivariate generating function $F(P, \sigma; x_1, x_2, ..., x_n)$ (denoted in short by $F(P, \sigma)$) in the variables $x_1, x_2, ..., x_n$ as

$$F(P, \sigma) = \sum_{f \in \alpha(P,\sigma)} x_1^{f(X_1)} x_2^{f(X_2)} ... x_n^{f(X_n)},$$

where $\alpha(P, \sigma)$ is the class of all (P, σ)-partitions. As before, by substituting all x_i's with q, we get the univariate or counting generating function $U(P, \sigma)$ of the number of (P, σ)-partitions:

$$U(P, \sigma) = \sum_{f \in \alpha(P,\sigma)} q^{f(X_1)+f(X_2)+...+f(X_n)}.$$

The coefficient of q^m in $U(P, \sigma)$ is equal to the number of (P, σ)-partitions of m. If σ is a natural labelling, we denote $F(P, \sigma)$ as $F(P)$ and $U(P, \sigma)$ as $U(P)$.

2.6 Informal definitions of #P and #P-complete classes

In the complexity class NP, a decision problem is in NP, if a candidate solution can be verified in polynomial time. Normally a problem in the class NP is of the form:"Are there any solutions that satisfy a given set of constraints?" (E.g. Are there any subsets of a set of integers that sum to k?) The corresponding #P problem is of the form: "How many solutions are there that satisfy a given set of constraints?" (E.g. How many subsets of that set of integers are there that sum to k?). The complexity class #P contains the set of counting problems associated with the corresponding decision problems in NP. A #P problem must be at least as hard as the corresponding NP problem, as other wise one can simply count the number of solutions and check if it is greater than zero.

A problem is #P-complete if it is in #P and every problem in #P can be reduced to it in polynomial time. It is believed that there are no polynomial time algorithms to solve #P-complete problems. Usually, for an NP-complete problem, counting the number of solutions is #P-complete. On the other hand, there are some problems in P whose corresponding counting problem is #P-complete.

2.6.1 Formal definitions of #P and #P-complete

A *counting Turing machine* is a standard non-deterministic TM which has an auxiliary output device that prints in binary notation, on a special tape, the number of accepting computations induced by the input. For most non-deterministic algorithms, each accepting computation corresponds to a solution to the problem. The (worst case) *time complexity* of a counting TM is $f(n)$, if the longest accepting computation induced by the set of all inputs of size n takes $f(n)$ steps. #P is the class of counting functions that can be computed by counting TMs of polynomial time complexity. An *oracle* TM is a TM that has a query tape, an answer tape and some working tapes and returns the answer to a query in unit time. The complexity class of problems solvable by an algorithm in class A with an oracle for a problem in class B is written as A^B. Let FP denote the class of functions computed by deterministic polynomial time TM. A problem y is #P-complete iff $\#P \subseteq FP^y$ and $y \in \#P$.

Chapter 3

Background of $2 \times 2 \times n$ solid partitions

In this chapter, we discuss the background of $2 \times 2 \times n$ solid partitions and explain in detail the problem that we will be solving.

3.1 The generating function for solid partitions

3.1.1 Plane partitions

A *plane partition* is a two dimensional extension of a partition. A $p \times n$ plane partition of m is a two-dimensional array of non-negative integers

$$\begin{array}{ccccc} m_{11} & m_{12} & m_{13} & \cdots & m_{1n} \\ m_{21} & m_{22} & m_{23} & \cdots & m_{2n} \\ m_{31} & m_{32} & m_{33} & \cdots & m_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{p1} & m_{p2} & m_{p3} & \cdots & m_{pn} \end{array}$$

such that

$$m_{ij} \geq m_{i+1j}, \;\; 1 \leq i < p, \; 1 \leq j \leq n$$

$$m_{ij} \geq m_{ij+1}, \;\; 1 \leq i \leq p, \; 1 \leq j < n$$

and

$$\sum_{i=1,j=1}^{i \leq p, \; j \leq n} m_{ij} = m.$$

For example, a 3×3 plane partition of 10 is

$$\begin{matrix} 3 & 2 & 0 \\ 2 & 1 & 0 \\ 2 & 0 & 0 \end{matrix} \cdot$$

The closed form of the generating function, $G_{p \times n}(q)$, of $p \times n$ plane partitions was given by MacMahon in [6] as:

$$G_{p \times n}(q) = \frac{1}{(q;q)_n (q^2;q)_n ... (q^p;q)_n}, \tag{3.1}$$

where

$$(q^i;q)_n = (1-q^i)(1-q^{i+1})...(1-q^{i+n-1}).$$

The coefficient of q^m is the number of $p \times n$ plane partitions of m. A $p \times n$ plane partition becomes an ordinary partition when $p = 1$ and the generating function for that is given by Equation 2.7.

3.1.2 Solid partitions

A *solid partition* is a three dimensional extension of a plane partition. A $h \times p \times n$ solid partition of m as defined in [7], is a representation of m as a solution to an equation of the form:

$$m = \sum_{i,j,k=1}^{i \leq h, \; j \leq p, \; k \leq n} m_{ijk}, \tag{3.2}$$

where m_{ijk} are non negative integers satisfying

$$m_{ijk} \geq m_{i+1jk}, \;\; 1 \leq i < h, \; 1 \leq j \leq p, \; 1 \leq k \leq n$$
$$m_{ijk} \geq m_{ij+1k}, \;\; 1 \leq i \leq h, \; 1 \leq j < p, \; 1 \leq k \leq n$$
$$m_{ijk} \geq m_{ijk+1}, \;\; 1 \leq i \leq h, \; 1 \leq j \leq p, \; 1 \leq k < n \quad .$$

A closed form for the generating function, $G_{h\times p\times n}(q)$, of $h \times p \times n$ solid partitions is not known. MacMahon in [6] conjectured the generating function, $G(q)$, of unbounded solid partitions (i.e. when $h, p, n \to \infty$) to be

$$G(q) = \prod_{k=1}^{\infty} (1-q^k)^{-k(k+1)/2}, \tag{3.3}$$

which was later proved to be incorrect [7, 8]. In particular, Equation 3.3 is incorrect for $n \geq 6$. The closed form representation of the generating function even for special cases, like $3 \times 2 \times n$ solid partitions or $3 \times 3 \times n$ solid partitions, is not known so far. In this thesis, our main goal is to derive a recurrence for the generating function, $G_{2\times 2\times n}(q)$, of one such special case defined by $2 \times 2 \times n$ solid partitions.

3.1.3 Solid partitions as a special case of P-partitions

Observe that solid partitions (and plane partitions) are just a special case of P-partitions defined in Section 2.5.2. When the cover relations of the $h \times p \times n$ poset, $P_{h\times p\times n}$, is given by the inequalities of $h \times p \times n$ solid partitions, the set of P-partitions of $P_{h\times p\times n}$, are same as $h \times p \times n$ solid partitions. Hence the generating function, $G_{h\times p\times n}(q)$, of $h \times p \times n$ solid partitions is the same as the generating function, $U(P_{h\times p\times n})$, of the set of P-partitions of $P_{h\times p\times n}$.

3.2 Counting linear extensions

Recall from Section 2.4.3, that a linear extension w is a total ordering of a poset that is consistent with the partial order. The number of linear extensions $\ell(P)$ of a poset P can vary from just 1 to as high as $n!$. For example, the total order poset defined in Figure 3.1(a) has just one linear extension. However, the poset in Figure 3.1(b), has no relations among the elements and thus it has $4! = 24$ linear extensions. To "list" all the linear extensions of a poset P would take time $O(\ell(P))$

Figure 3.1: (a) A poset of complete order (b) A poset with no order

which could be exponential in the number of elements of P in the worst case. However, if we are interested in just counting the number of linear extensions, we might expect an easier polynomial time algorithm. But this is not the case. Brightwell and Winkler in 1991 [9] proved that the problem of counting linear extensions belongs to the complexity class #P-complete defined in Section 2.6.

The problem of counting the linear extensions is in #P as we can check in polynomial time, whether a given linear extension w is consistent with the poset P. The problem was proved in [9] to be #P-complete by proving that if an oracle Turing machine could count the linear extensions of the poset P in polynomial time, then it could also count the number of satisfactory assignments to an instance of 3-SAT in polynomial time. The corresponding decision problem is in the complexity class P, i.e., we can always tell if a poset has a linear extension (as every non-empty poset has at least one linear extension).

3.2.1 Linear extensions and P-partitions

The linear extensions of a poset P are related to the set of P-partitions of P, through the fundamental lemma of P-partitions. Let π be a permutation of the set $X_n = \{1, 2, 3, ..., n\}$. Define $A(\pi)$ to be the set of all functions $f : X_n \rightarrow \{1, 2, 3, ...\}$ such that

$$f(\pi(1)) \leq f(\pi(2)) \leq ... \leq f(\pi(n))$$

and whenever $\pi(s) > \pi(s+1)$, $f(\pi(s)) < f(\pi(s+1))$ [10, 11]. For example, if $\pi = (2, 4, 1, 3)$, $A(\pi)$ is the infinite set of all maps $f : \{1, 2, 3, 4\} \rightarrow \{1, 2, 3, ...\}$ such that $f(2) \leq f(4) < f(1) \leq f(3)$.

Now, the *fundamental lemma of P-partitions* [5] says that if $A(P)$ represents the set of all P-partitions of a poset P , then $A(P)$ is the disjoint union of the sets $A(\pi)$ over all linear extensions π of P:

$$A(P) = \coprod_{\pi \in L(P)} A(\pi).$$

Counting linear extensions from generating function of the set of P-partitions

In terms of generating functions, the single variable generating function $U(P)$ (defined at the end of Section 2.5.3) of the set of P-partitions of a poset $P = (X, \leq_P)$ can be written as:

$$U(P) = \frac{W(q)}{(1-q)(1-q^2)...(1-q^{|X|})}, \tag{3.4}$$

where $W(q)$ is a polynomial in q with integer coefficients, called the *W-polynomial* of the poset P [5, 12], satisfying

$$\lim_{q \to 1} W(q) = \ell(P). \tag{3.5}$$

Also, the W-polynomial can be calculated as

$$W(q) = \sum_{w \in L(P)} q^{ind(w)} \tag{3.6}$$

where $ind(w) = \{\sum j | x_j > x_{j+1}\}$ for $x_j \in w$.

Thus the linear extensions of a poset can be counted from the generating function of the set of P-partitions of the poset. We noted in Section 3.1.3 that the generating function of solid partitions and plane partitions is same as the generating function of the set of P-partitions of the corresponding poset. Hence, we can compute $\ell(P)$ of the corresponding poset from the generating function of solid partitions and plane partitions.

3.2.2 Counting linear extensions for posets represented by plane partitions and solid partitions

The $p \times n$ plane partition defined in Section 3.1.1 has a closed form for the generating function, $G_{p \times n}(q)$, which is same as the single variable generating function, $U(P_{p \times n})$, of the set of P-partitions of the corresponding poset, $P_{p \times n}$. Using Equations 3.4 and 3.5, we get the formula to count the number of linear extensions, $\ell(P_{p \times n})$, of $P_{p \times n}$ as :

$$\begin{aligned}
\ell(P_{p\times n}) &= \lim_{q\to 1} U(P_{p\times n}) * \prod_{i=1}^{pn}(1-q^i) \\
&= \lim_{q\to 1} \frac{(1-q)(1-q^2)(1-q^3)...(1-q^{pn})}{(q;q)_n(q^2;q)_n...(q^p;q)_n} \\
&= \frac{(pn)!}{\frac{n!}{0!}\frac{(n+1)!}{1!}\frac{(n+2)!}{2!}...\frac{(n+p-1)!}{(p-1)!}} \\
&= \frac{0!1!2!...(p-1)!(pn)!}{(n!)(n+1)!(n+2)!...(n+p-1)!}.
\end{aligned}$$

However, for $h\times p\times n$ solid partitions neither a closed form for the generating function, nor a formula to count the number of linear extensions of the poset represented by them is known. Another goal of this thesis is to compute the number of linear extensions, $\ell(P_{2\times 2\times n})$, of the $2\times 2\times n$ poset, $P_{2\times 2\times n}$, from the generating function of $2\times 2\times n$ solid partitions.

3.3 Our Problem

Our focus in this thesis, is on the $2\times 2\times n$ solid partitions. They are defined by Equation 3.2 with $h=2$ and $p=2$. We will derive an explicit recurrence for the generating function, $G_{2\times 2\times n}(q)$, of $2\times 2\times n$ solid partitions. We will use a set of digraph rules that will be described in Section 4.2 to derive this recurrence.

Once we have the generating function, we will use Equations 3.4 and 3.5 to compute the number of linear extensions, $\ell(P_{2\times 2\times n})$, of $P_{2\times 2\times n}$.

Chapter 4

The main technique

As discussed in Section 3.3, our problem is to compute a recurrence for the generating function of $2 \times 2 \times n$ solid partitions. We will make use of a set of rules called the "digraph methods" [2] to build this recurrence. These digraph methods are derived from a set of five rules called the "five guidelines" [3]. Both the digraph methods and the five guidelines treat partitions as a set of integer solutions to linear inequalities C as described in Section 2.3.1.

4.1 Five guidelines

Corteel, Lee and Savage [3] proposed the "five guidelines", which focused on getting a recurrence for the generating function for the set of solutions to a set of linear inequalities C. These techniques were built based on *partition analysis* techniques developed by MacMahon and were successful in producing recurrences for many well known problems. The main advantage of getting a recurrence for a generating function F_C is not only to be able to write a program for F_C, but also to get a closed form of the generating function for the infinite family. It is also proved that these set of five guidelines are sufficient for any system of homogeneous linear inequalities with integer coefficients. A brief overview of the five guidelines is presented below.

Theorem 4.1 *Let $S = (s_1, s_2, ..., s_n)$ be a sequence that satisfies the set of linear constraints C where each constraint $c \in C$ of the form*

$$c : [a_0 + \sum_{i=1}^{n} a_i s_i \geq 0],$$

where a_i's are integers (possibly negative).

1. *If $s_1 \geq t$ for some integer $t \geq 0$, is the only constraint in C, then*

$$F_C = \frac{x_1^t}{(1-x_1)}.$$

2. *If C_1 is a set of constraints on the variables $s_1, s_2, ..., s_j$ and C_2 is the set of constraints on the variables $s_{j+1}, s_{j+2}, ..., s_n$, then*

$$F_{C_1 \cup C_2}(x_1, x_2, ..., x_n) = F_{C_1}(x_1, x_2, ..., x_j) F_{C_2}(x_{j+1}, x_{j+2}, ..., x_n).$$

3. *If the set of constraints C on variables $s_1, s_2, ..., s_n$ (containing the constraint $s_i \geq 0, 1 \leq i \leq n$) implies the constraint $s_i - a s_j \geq 0$, for any integer a, then*

$$F_C(X_n) = F_{C_{s_i \leftarrow s_i + a s_j}}(X_n; x_j \leftarrow x_j x_i^a).$$

4. *If c is any constraint with the same variables as C, then*

$$F_C(X_n) = F_{C \cup \{c\}}(X_n) + F_{C \cup \{\neg c\}}(X_n).$$

5. *If $c \in C$, then*

$$F_C(X_n) = F_{C - \{c\}}(X_n) - F_{C - \{c\} \cup \{\neg c\}}(X_n).$$

Here X_n denotes $x_1, x_2, ..., x_n$ and $\neg c$ is the negation of c given by $-a_0 - \sum_{i=1}^{n} a_i s_i \geq 1$, where $c \in C$.

4.2 Digraph methods

There are certain types of constraints C, "$s_a \geq s_b$" and "$s_a > s_b$" for which we can simplify the procurement of a recurrence for the generating function. These constraints can be represented as a directed graph, where the vertices corresponding to variables $(s_1, s_2, ...s_n)$ are labeled $1, 2, 3, ..., n$ and there is an edge (or strict edge) between i and j if C has the constraint $s_i \geq s_j$ (or $s_i > s_j$ respectively). The generating function of such constraints can be obtained from a set of digraph methods proposed by Davis, D'Souza, Lee and Savage [2] which is based on the five guidelines

explained in Section 4.1. A number of integer partitions problems, when solved using the digraph rules have given a simpler and neater solution.

Let $G(V, E)$ be a directed graph with vertices $V = \{1, 2, ..., n\}$ and edges E where some edges are designated as *strict*. Let S_C be the set of non-negative integer sequences $S = (s_1, s_2, ..., s_n)$ that satisfy the constraints $s_i \geq s_j$ and $s_i > s_j$. We would like to compute the multivariate generating function

$$F_G(x_1, x_2, ..., x_n) = \sum_{s \in S_C} x_1^{s_1} x_2^{s_2}, ..., x_n^{s_n}.$$

Theorem 4.2 *For $v_i \in V$, let G' denote the graph obtained from G by adding a vertex v_{n+1} and an edge from v_{n+1} to v_i. Then*

$$F_{G'}(x_1, ..., x_n, x_{n+1}) = \frac{F_G(x_1, ..., x_{i-1}, x_i x_{n+1}, x_{i+1}, ..., x_n)}{(1 - x_{n+1})}$$

If the inequality corresponding to the edge (v_{n+1}, v_i) is strict, then the generating function on the right hand side is multiplied by x_{n+1}. Figure 4.1 shows the an incoming edge (v_{n+1}, v_i).

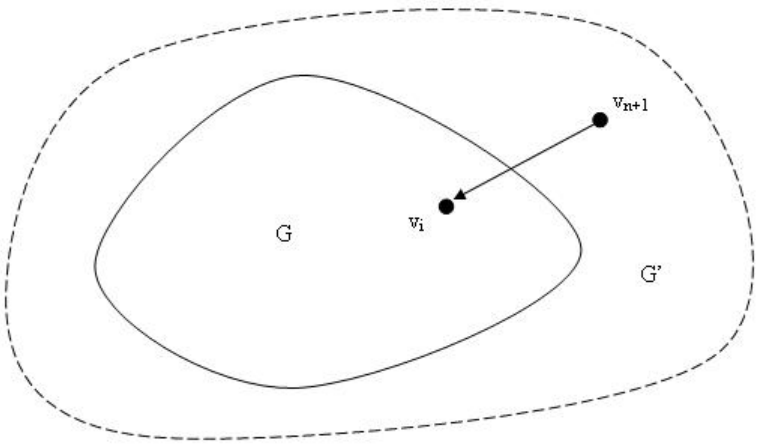

Figure 4.1: G along with the incoming edge forming G'

Below we present a set of five rules that we will be using in our thesis to solve our problem. Some of these are derived from Theorems 4.1 and 4.2 while some are just graphical representations of the two theorems (most of them can be found in [13]). An edge is represented as usual whereas, a strict edge is represented with a double head as shown in Figure 4.2.

Rule 1: Independent Vertex For a graph $G(V, E)$, where $V = v_1, v_2, ..., v_n$, if we know the generating function F_G, then the generating function $F_{G'}$ of the graph defined by $G'(V', E)$ where

Figure 4.2: (a) Normal edge representing $s_1 \geq s_2$ (b) Strict edge representing $s_3 > s_4$

$V' = V \cup \{v_{n+1}\}$ is given by

$$F_{G'}(x_1, x_2, ..., x_n, x_{n+1}) = \frac{F_G(x_1, x_2, ..., x_n)}{(1 - x_{n+1})}.$$

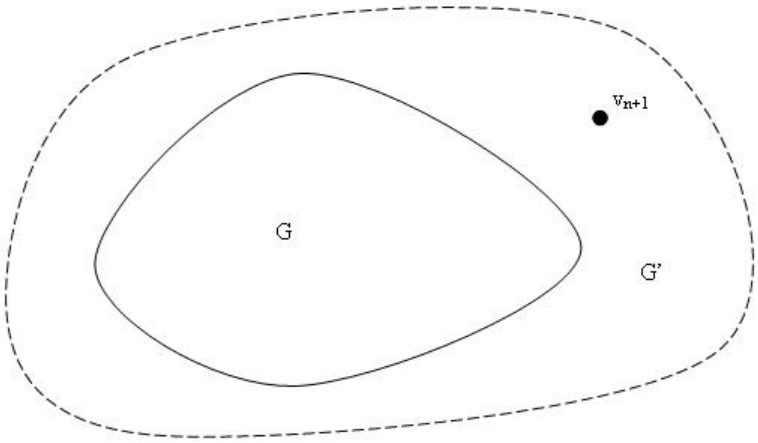

Figure 4.3: G' is the graph G along with the independent vertex v_{n+1}

Figure 4.3 shows a graph G' formed from G and an independent vertex v_{n+1}. The generating function of G' can be computed from the generating function of G. This rule can be derived from Theorem 4.1 (1) and Theorem 4.1 (2). A single vertex graph (G_1) with the vertex v_{n+1} represents only the constraint $s_{n+1} \geq 0$ and Theorem 4.1 (1) directly gives us the generating function of this graph as

$$\frac{1}{(1 - x_{n+1})}.$$

Theorem 4.1 (2) can then be used to combine the graphs G and G_1 to get the generating function of the graph G'.

Rule 2: Redundant Edge A constraint $s_i \geq s_j$ is redundant if it can be derived from two or more existing constraints. An edge that represents such a constraint is called a *redundant edge*. The redundant edge operator allows us to remove a redundant edge from a graph without modifying the generating function i.e. if G is a graph with a redundant edge between (v_i, v_k), and G' is the graph with the redundant edge removed, then

$$F_{G'}(x_1, x_2, ..., x_n) = F_G(x_1, x_2, ..., x_n)$$

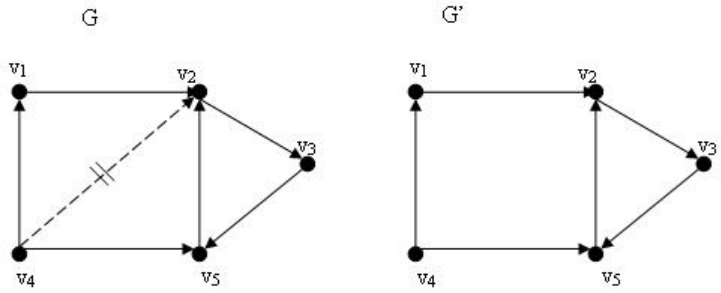

Figure 4.4: G' has the redundant edge removed

Figure 4.4 shows a graph G with a redundant edge (v_4, v_2). On removing it, we get the graph G' whose generating function remains unaltered. This is not directly derived from any of the theorems, but intuitively follows from the definition of a redundant constraint.

Rule 3: Incoming Edge This is exactly Theorem 4.2.

Rule 4: Inclusion-Exclusion (1) The first rule is a direct consequence of Theorem 4.1 (4). Considering the constraint c to be of the form $s_i \geq s_j$, $\neg c$ becomes $s_i < s_j$. Translating to graphs, the generating function of a graph G can be obtained from the generating function of graphs G_1 and G_2 as:

$$F_G(x_1, x_2, ..., x_n) = F_{G_1}(x_1, x_2, ..., x_n) + F_{G_2}(x_1, x_2, ..., x_n)$$

where G represents the set of constraints C, G_1 represents $C \cup \{c\}$ and G_2 represents $C \cup \{\neg c\}$.

Here c and $\neg c$ are constraints on variables that exist in C. Figure 4.5 shows a graphical representation of this rule.

Rule 5: Inclusion-exclusion (2) The second inclusion-exclusion rule is a direct consequence of Theorem 4.1 (5). As before, if we consider constraint c to be of the form $s_i \geq s_j$, $\neg c$ becomes $s_i < s_j$. Translating to graphs, we can get the generating function of a graph G from graphs G' and G'' as:

$$F_G(x_1, x_2, ..., x_n) = F_{G'}(x_1, x_2, ..., x_n) - F_{G''}(x_1, x_2, ..., x_n)$$

where G represents the set of constraints C, G' represents $C - \{c\}$ and G'' represents $C - \{c\} \cup \{\neg c\}$. Figure 4.6 shows a graphical representation of this rule. It can be seen that the second inclusion-exclusion principle is just a rearrangement of the first inclusion-exclusion principle, nevertheless, we shall mention it as a separate rule to reference it easily.

These set of five rules derived from the two theorems are proved to be sufficient to derive the generating function of any set of sequences described by inequalities of the form $s_a \geq s_b$ and $s_a > s_b$ [13].

4.3 Applicability of digraph rules to $2 \times 2 \times n$ solid partitions

We can compute the generating function of $2 \times 2 \times n$ solid partitions using the digraph methods, since they can be represented by inequalities of the form $s_a \geq s_b$. We can model them as a digraph as described in Section 4.2. The resulting digraph G_n is shown in Figure 4.7.

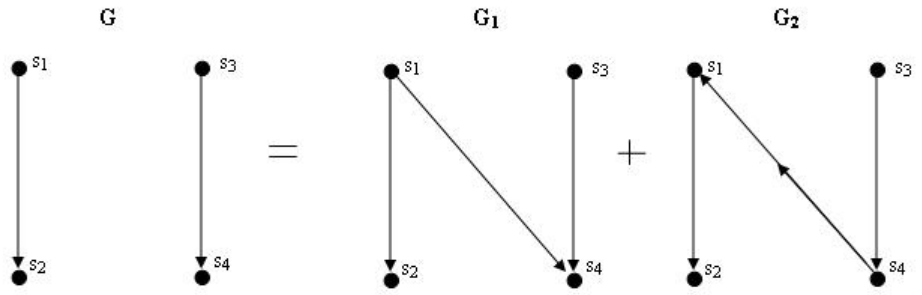

Figure 4.5: First Inclusion-exclusion principle

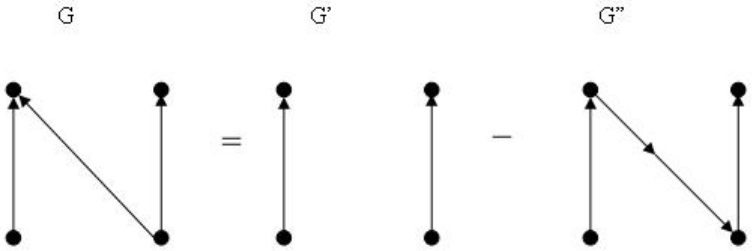

Figure 4.6: Second Inclusion-exclusion principle

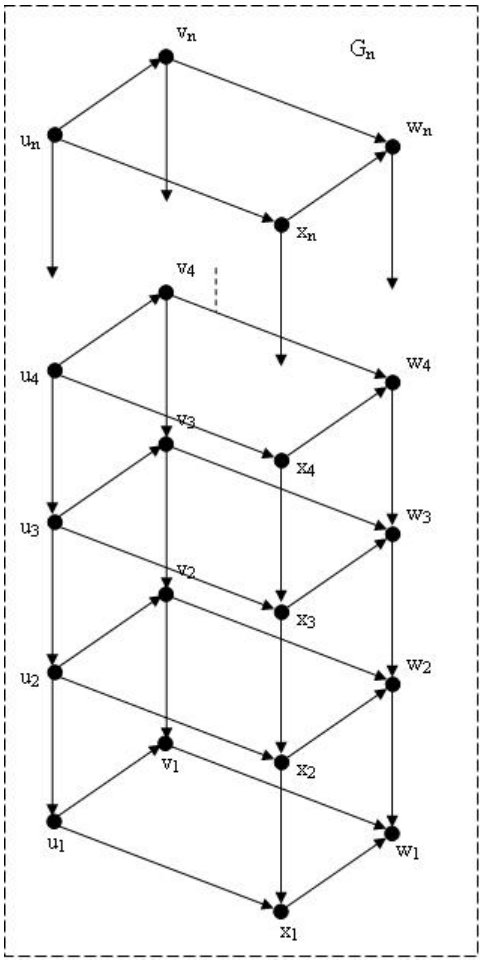

Figure 4.7: The digraph representing $2 \times 2 \times n$ solid partitions

Chapter 5

Derivation of recurrence

In this chapter, we will apply the digraph rules of Section 4.2 on $2 \times 2 \times n$ solid partitions (represented by the digraph G_n in Figure 4.7), to decompose it and derive a recurrence for its generating function.

5.1 Steps for decomposing G_n using digraph rules

We modeled our $2 \times 2 \times n$ solid partitions as the digraph G_n in Section 4.3 and now apply the digraph methods on it to decompose it. For the rest of this thesis, generating function of a graph means the generating function of the solid partitions represented by it.

Step 1: In the first step, we use Rule 4 from Section 4.2 on the graph (G_n) to generate two new edges (x_n, v_n) and (v_n, x_n) (which is strict). Figure 5.1 shows this process. It results in the two graphs H_n and H'_n.

As we can see in Figure 5.1, some edges are redundant in the resultant graphs ((u_n, v_n), (x_n, w_n) in H_n and (v_n, w_n), (u_n, x_n) in H'_n) and we can remove them without modifying the generating function according to Rule 2 from Section 4.2. We notice that H_n and H'_n are isomorphic to each other except that H'_n has a strict edge. So, most of the operations that we do on H_n would be repeated in the case of H'_n. Steps 2-10 will be breaking down of H_n whereas steps 11-20 would do the same to break down H'_n.

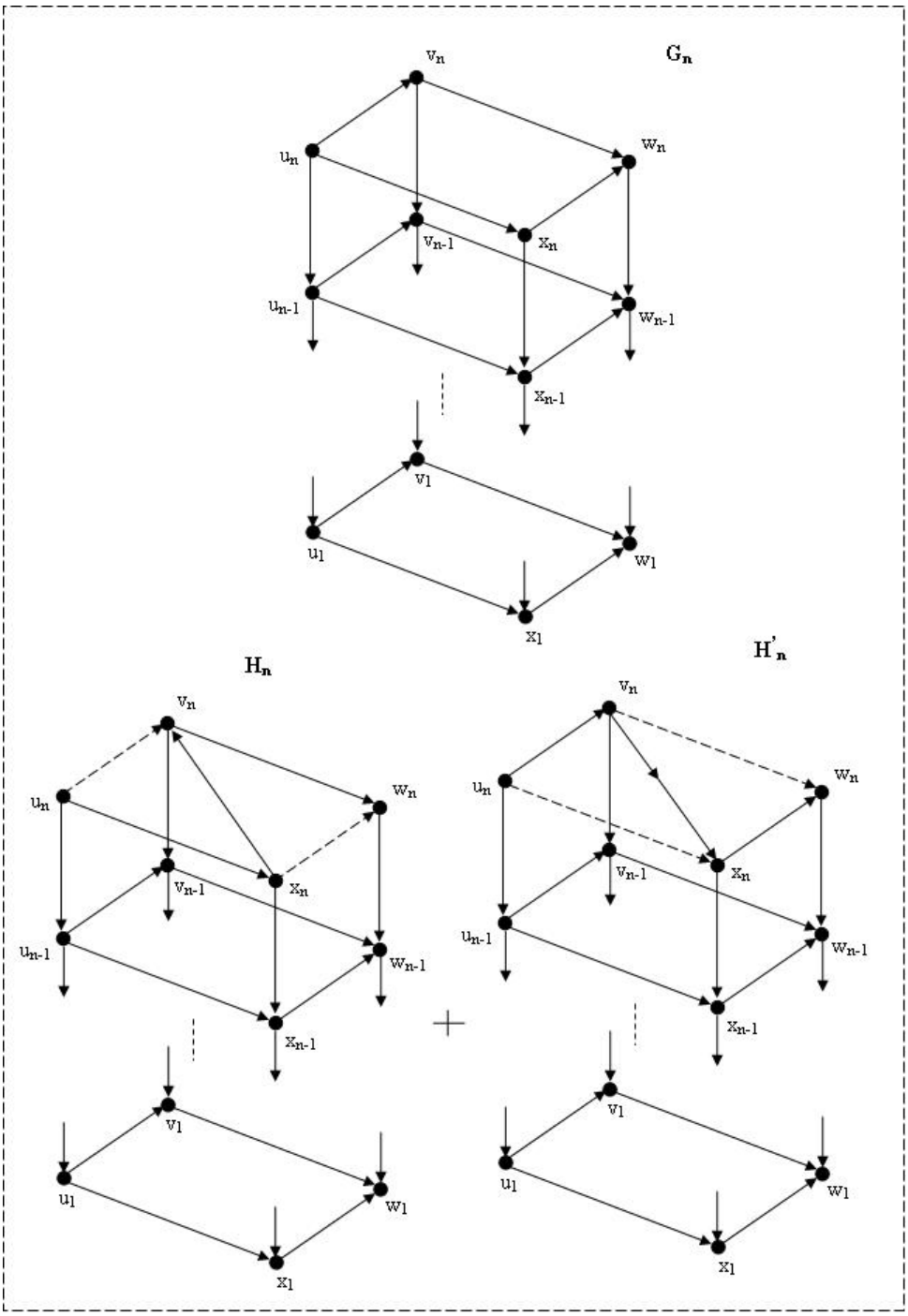

Figure 5.1: Step 1 of Decomposition

Step 2: In the second step, we take H_n and apply Rule 5 on the edge (x_n, x_{n-1}). This results in two graphs K_n and P_n as shown in Figure 5.2.

It can be seen that the edge (u_n, x_n) is redundant in P_n and can be removed from the graph without affecting the generating function as per Rule 2.

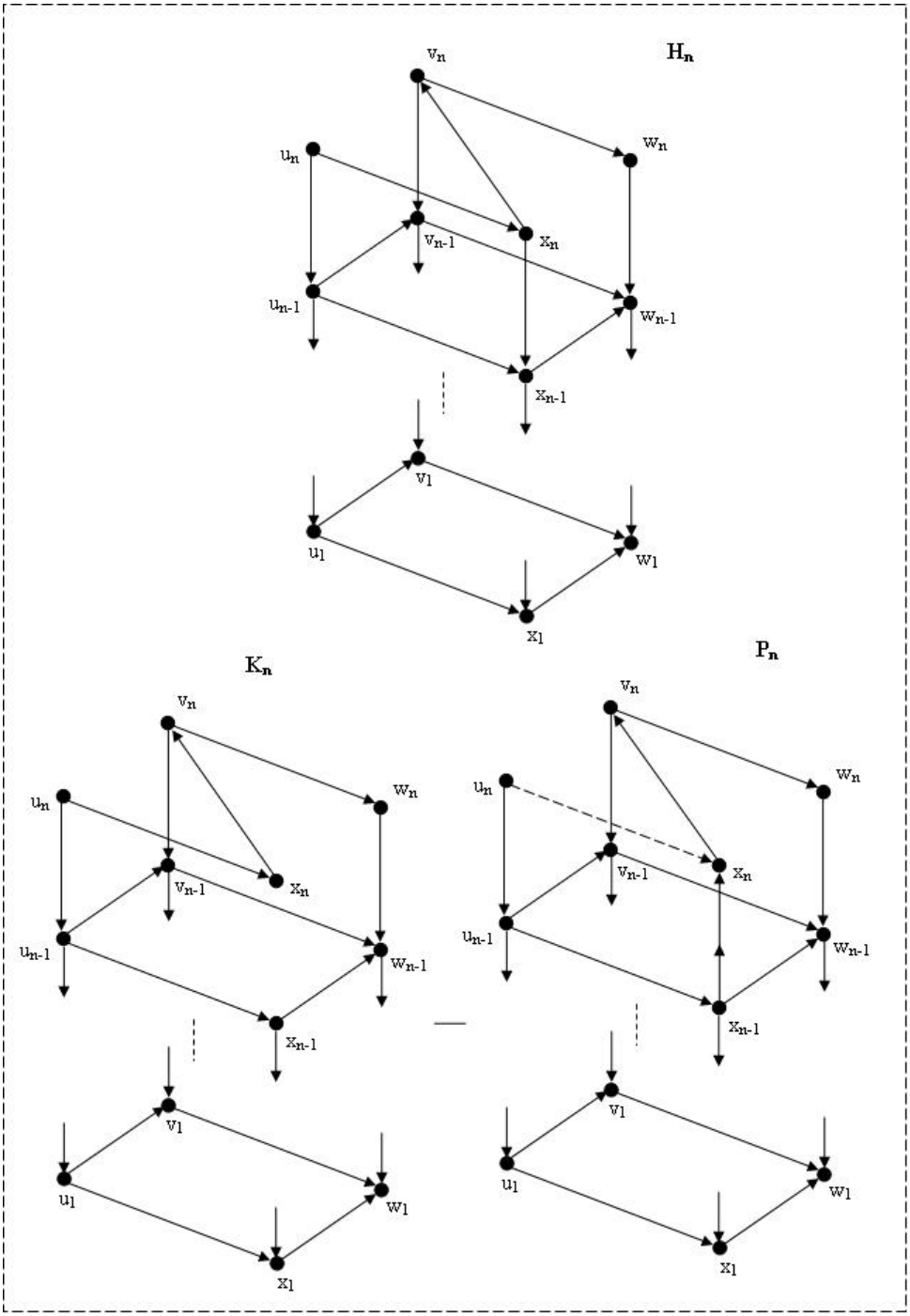

Figure 5.2: Step 2 of Decomposition

Step 3: In the third step, we take the graph P_n and apply rule 5 on the edge (w_n, w_{n-1}). We get the resultant graphs R_n and S_n as shown in Figure 5.3.

It can be seen that the edge (v_n, w_n) becomes redundant in the graph S_n. We now describe how both of these graphs can be reduced to D_n, shown in Figure 5.4.

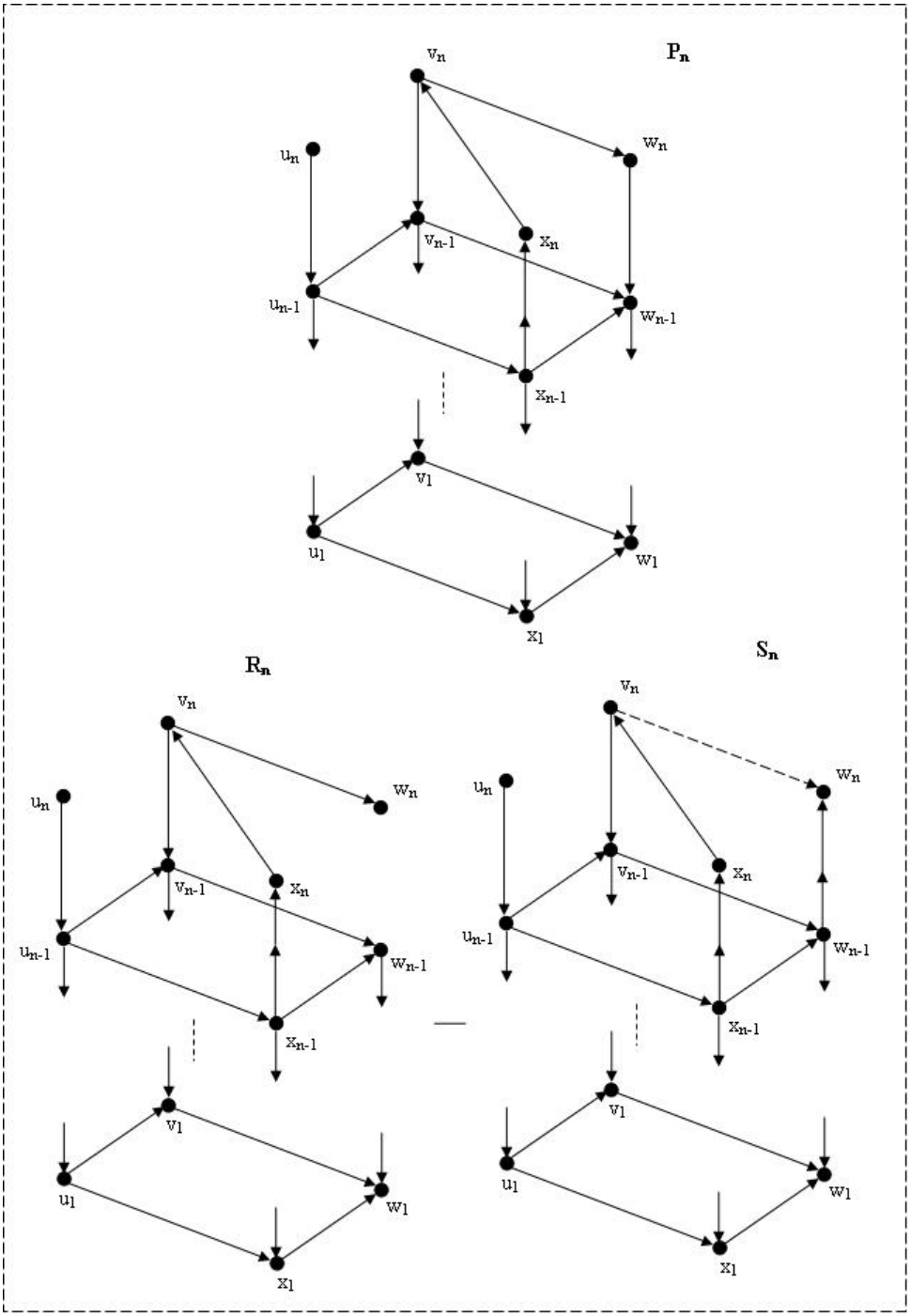

Figure 5.3: Step 3 of Decomposition

D_n can be derived from R_n by applying Rule 3 on the edge (u_n, u_{n-1}). On the edge (n_n, w_n), Rule 5 is applied, and to the resultant 2 graphs, Rule 3 and Rule 1 are applied. On doing all these, we get the graph D_n in each of these cases. Similarly D_n can be derived from S_n using the Rule 3, Rule 5 and Rule 1.

Next, to decompose D_n, we first use Rule 3, but here, not to remove a redundant edge but to add a redundant edge to the graph. We add the edge (x_{n-1}, v_{n-1}) which is redundant because the edges $(x_{n-1}, x_n), (x_n, v_n), (v_n, v_{n-1})$ imply that constraint (Figure 5.5).

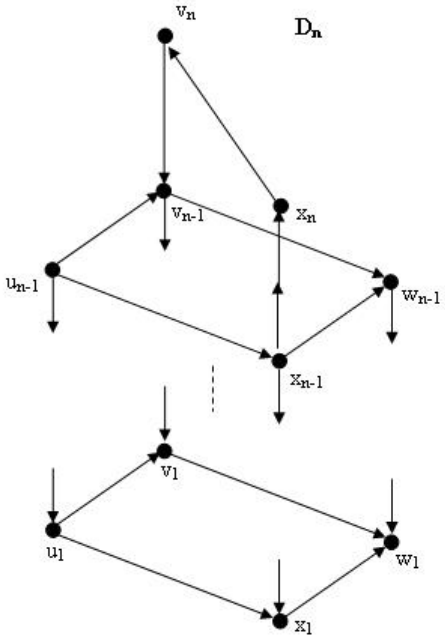

Figure 5.4: The Graph D_n

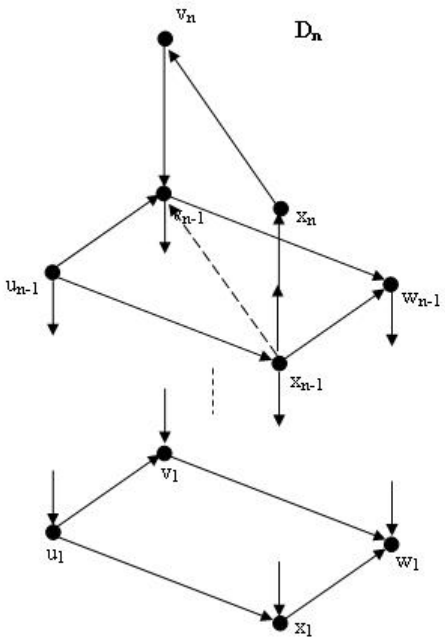

Figure 5.5: The Graph D_n with the redundant edge

Step 4: In the next step, we take D_n with the redundant edge and apply Rule 5 on the edge (x_{n-1}, x_n) and we get the two graphs T_n and W_n as shown in Figure 5.6.

The graph T_n can be reduced using Rule 3 twice to get H_{n-1}. As for the graph W_n, it can be further reduced as in step 5.

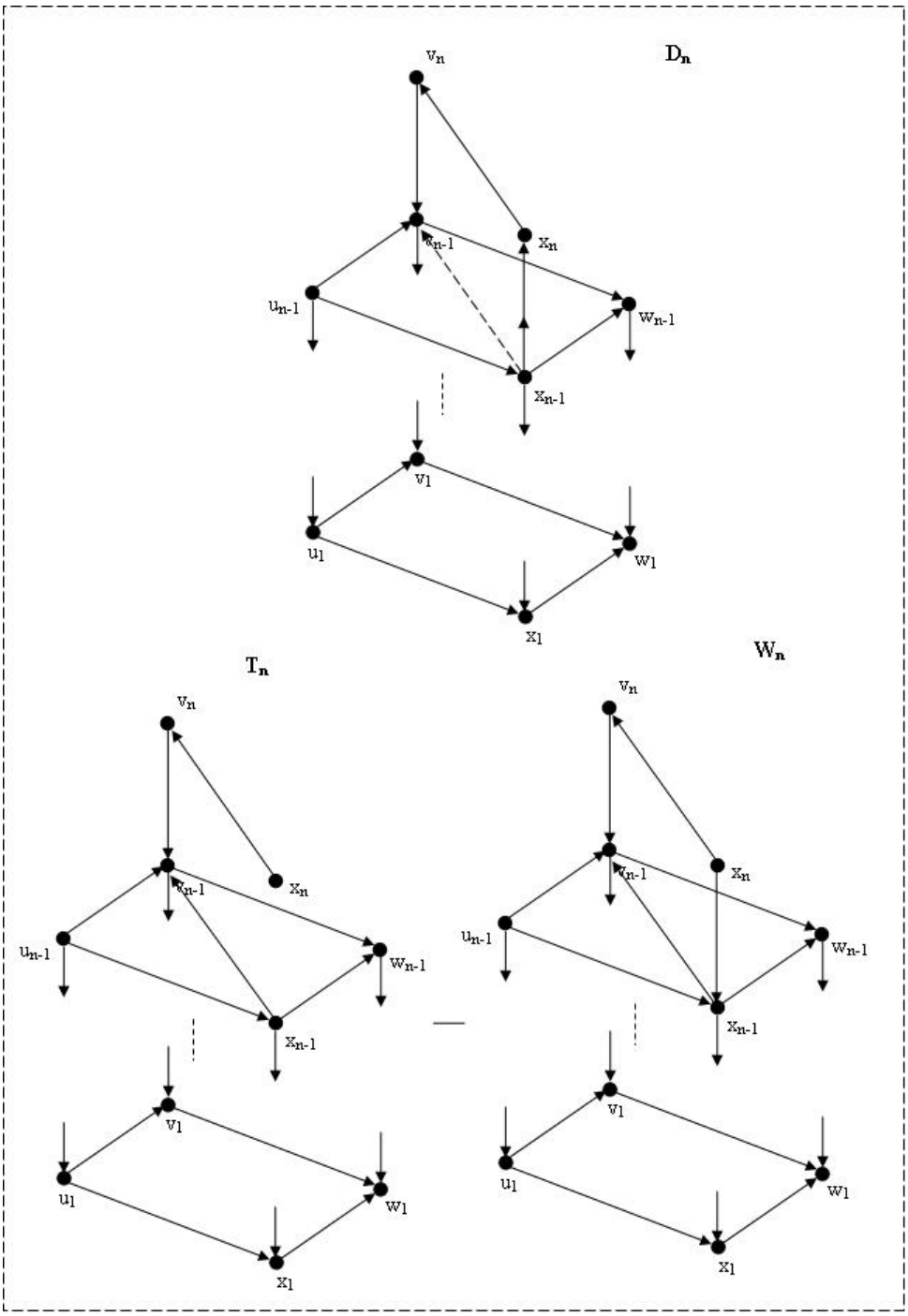

Figure 5.6: Step 4 of Decomposition

Step 5: In this step we take the graph W_n and apply Rule 5 on the edge (x_n, v_n). We get the two graphs U_n and V_n as shown in Figure 5.7.

The edge (v_n, v_{n-1}) becomes redundant in the graph V_n and hence can be discarded. Both U_n and V_n can be reduced to the graph H_{n-1} using Rule 3 on the two extra edges. We have reduced the graph P_n from step 2 to H_{n-1}. Now we take up the graph K_n and reduce it.

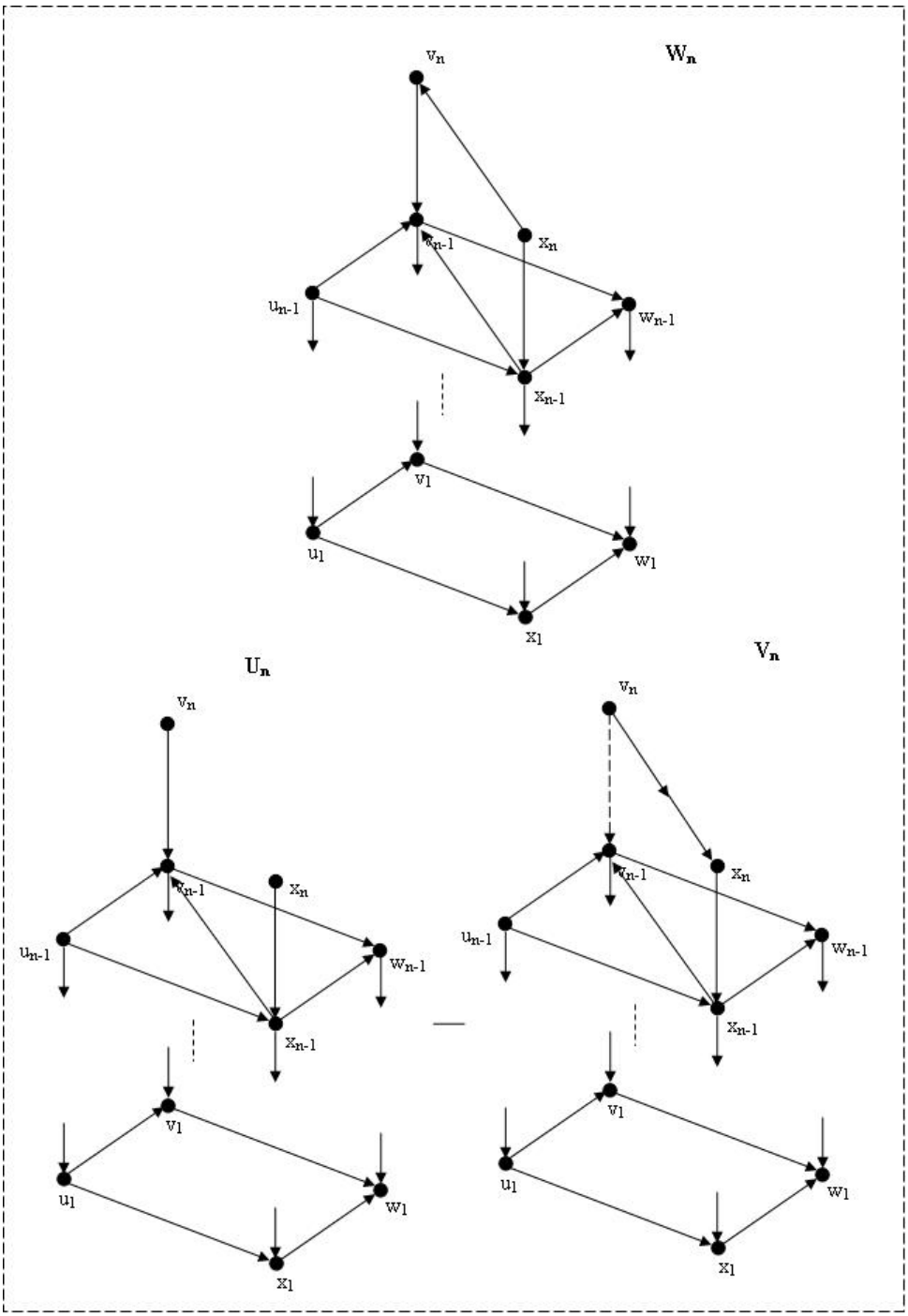

Figure 5.7: Step 5 of Decomposition

Step 6: In this step, we first use Rule 5 on the edge (w_n, w_{n-1}). We get the two graphs K_n^1 and K_n^2 as shown in Figure 5.8. It can be seen that the edge (v_n, w_n) is redundant in the graph K_n^2 and can be removed.

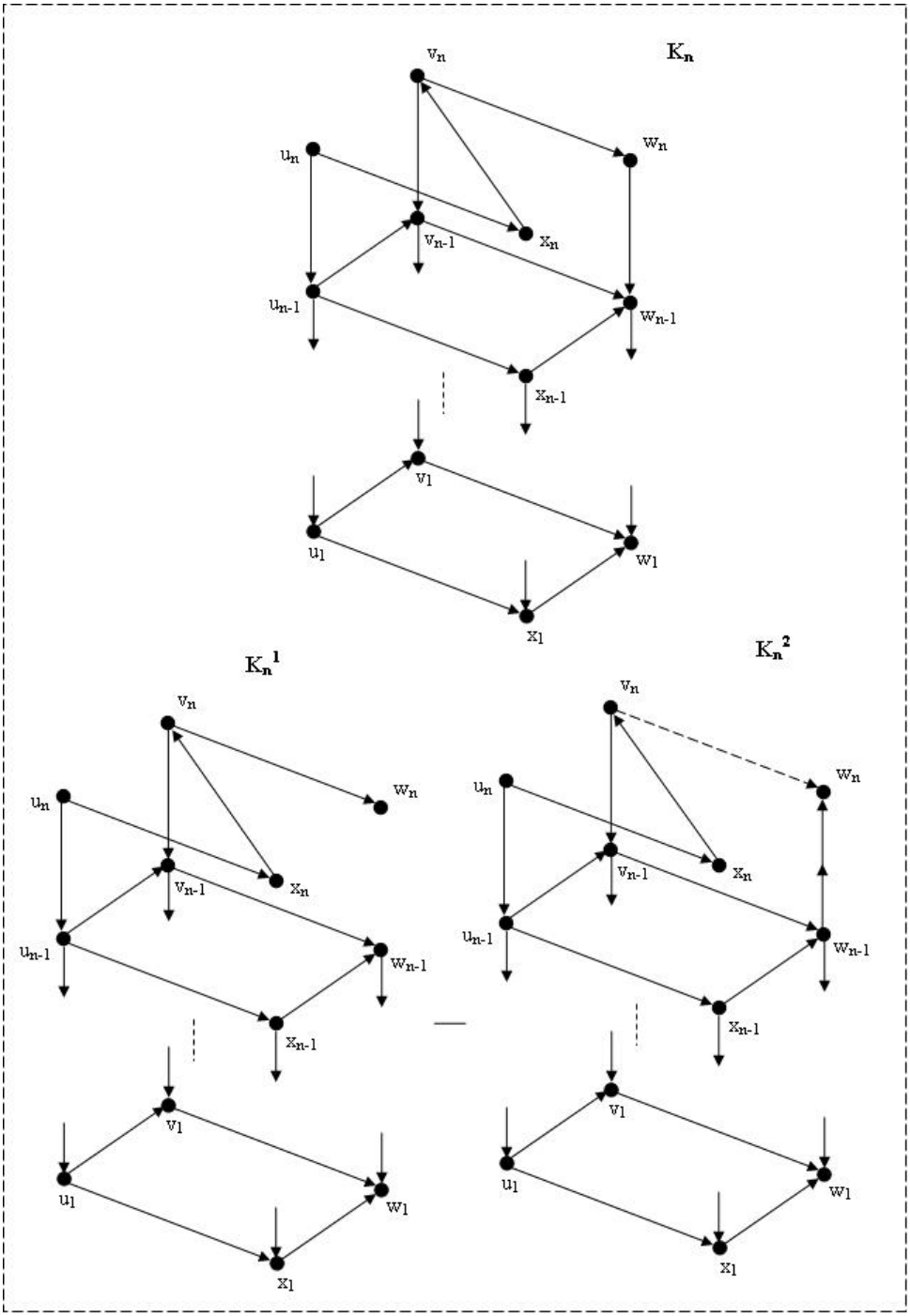

Figure 5.8: Step 6 of Decomposition

Step 7: In the next step, we take the graph K_n^1 and apply Rule 5 on the edge (u_n, x_n). We get the two graphs K_n^3 and K_n^4 as shown in Figure 5.9.

The graph K_n^3 can be reduced to G_{n-1} with further simplifications that will be shown later. We next, reduce the graph K_n^4.

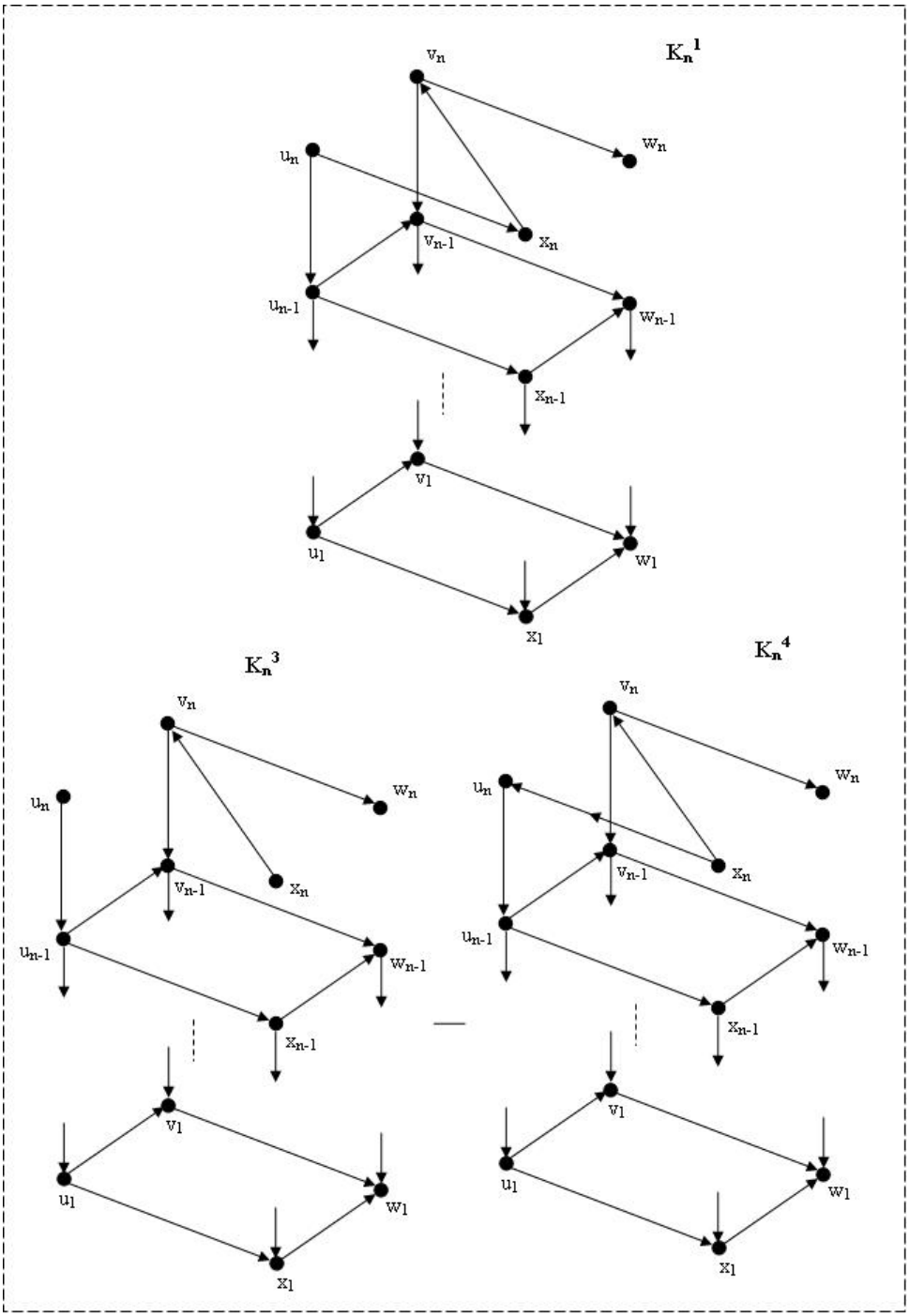

Figure 5.9: Step 7 of Decomposition

Step 8: In this step, we take the graph K_n^4, and use Rule 5 on the edge (x_n, v_n). We get the two graphs K_n^5 and K_n^6 as shown in Figure 5.10.

The edge (v_n, v_{n-1}) becomes redundant in the graph K_n^6. Both the graphs K_n^5 and K_n^6 can be reduced to G_{n-1} which will be shown later. We have reduced the graph K_n^1 from step 6 and now we have to reduce the graph K_n^2.

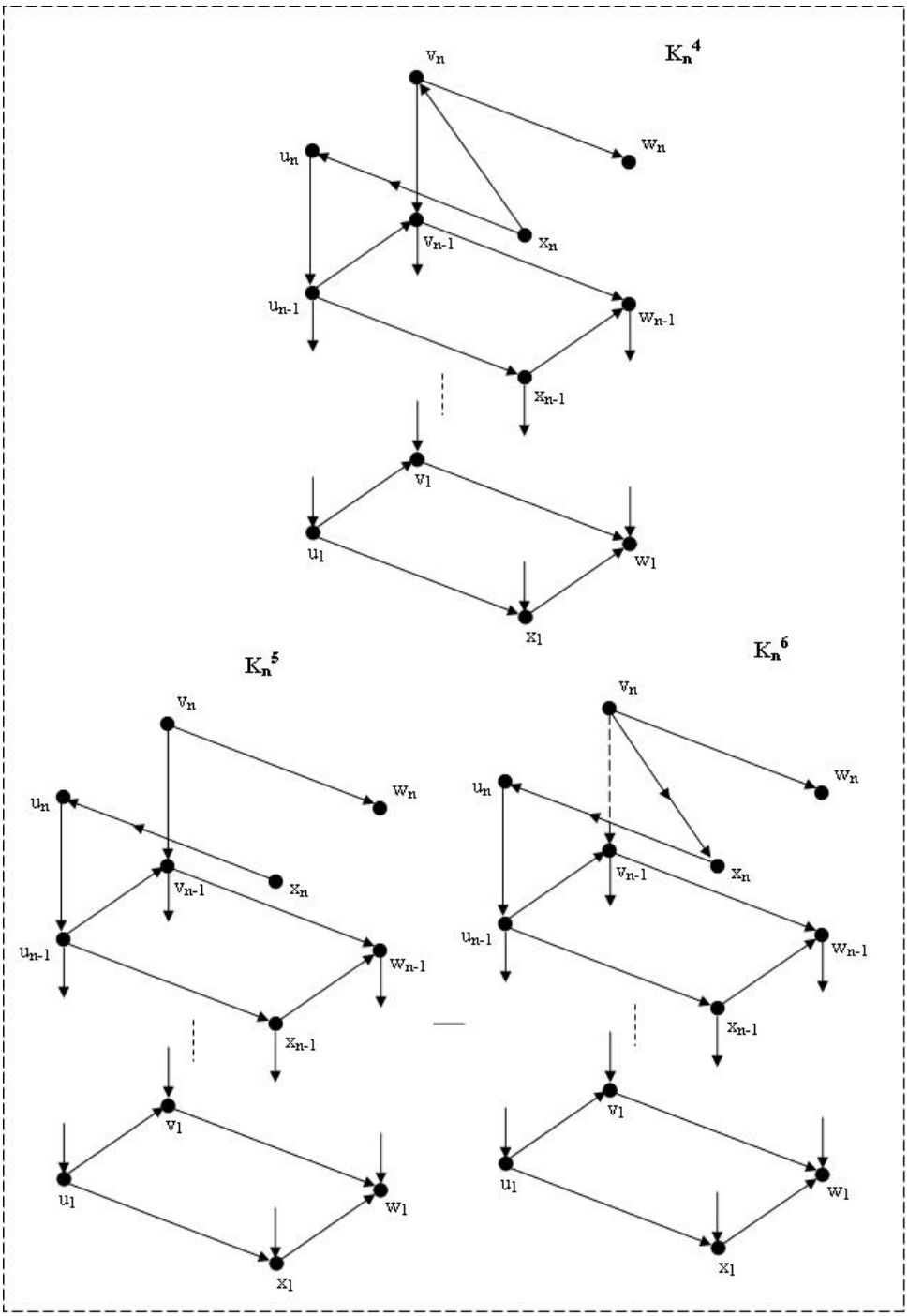

Figure 5.10: Step 8 of Decomposition

Step 9: In this step, we take the graph K_n^2 and use Rule 5 on the edge (u_n, x_n) and we get the 2 graphs K_n^7 and K_n^8 (Figure 5.11). The graph K_n^7 can be reduced to G_{n-1} which will be shown later. The graph K_n^8 has to be further simplified.

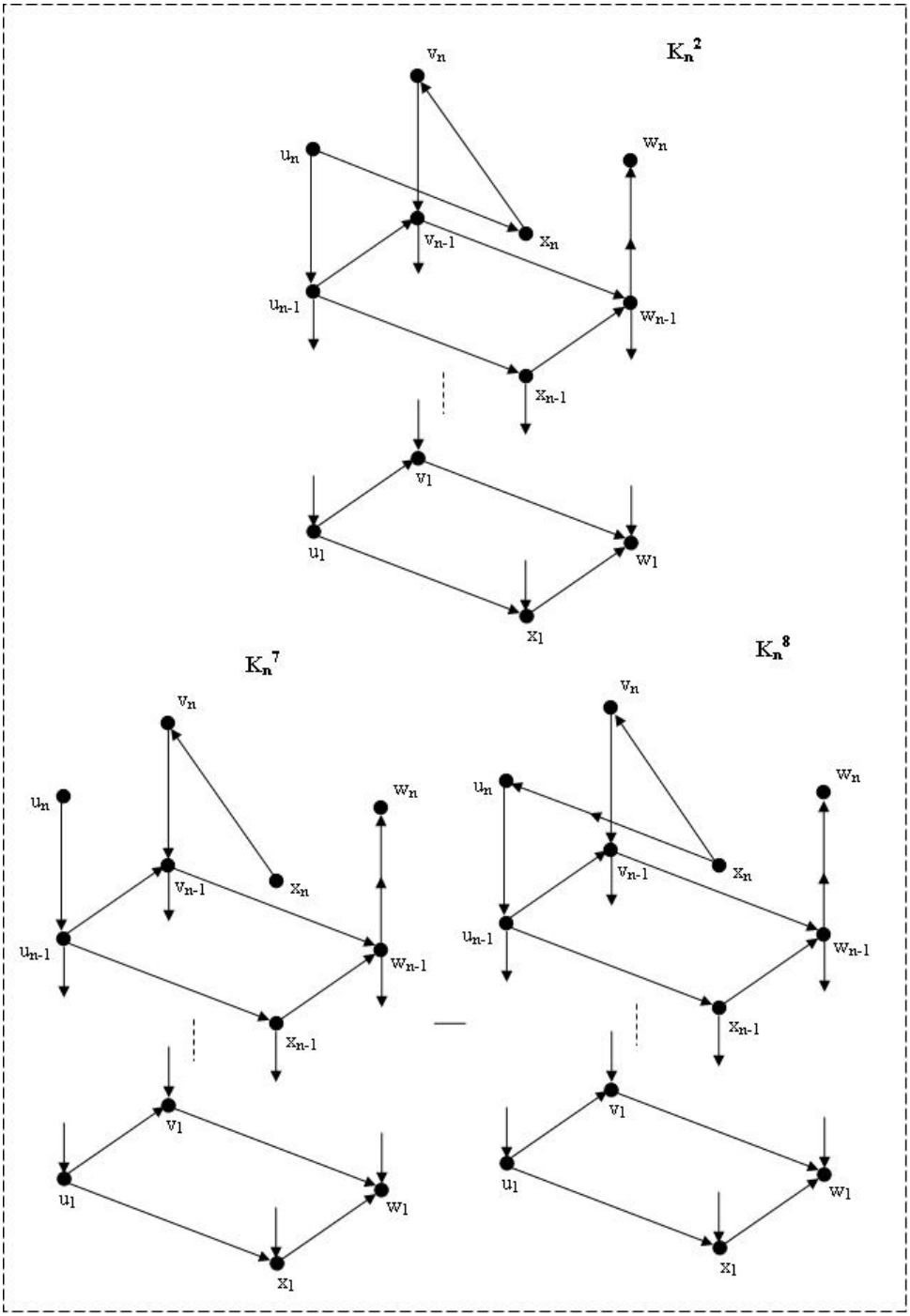

Figure 5.11: Step 9 of Decomposition

Step 10: In this step, we take the graph K_n^8 and apply Rule 5 on the edge (x_n, v_n) and we get the two graphs (K_n^9) and (K_n^{10}) as shown in the Figure 5.12.

Both K_n^9 and K_n^{10} can be reduced to G_{n-1} which will be shown later. We have now reduced H_n from step 1 into 6 instances of G_{n-1} and 6 instances of H_{n-1}. Now we have to reduce H'_n from step 1 in a similar manner.

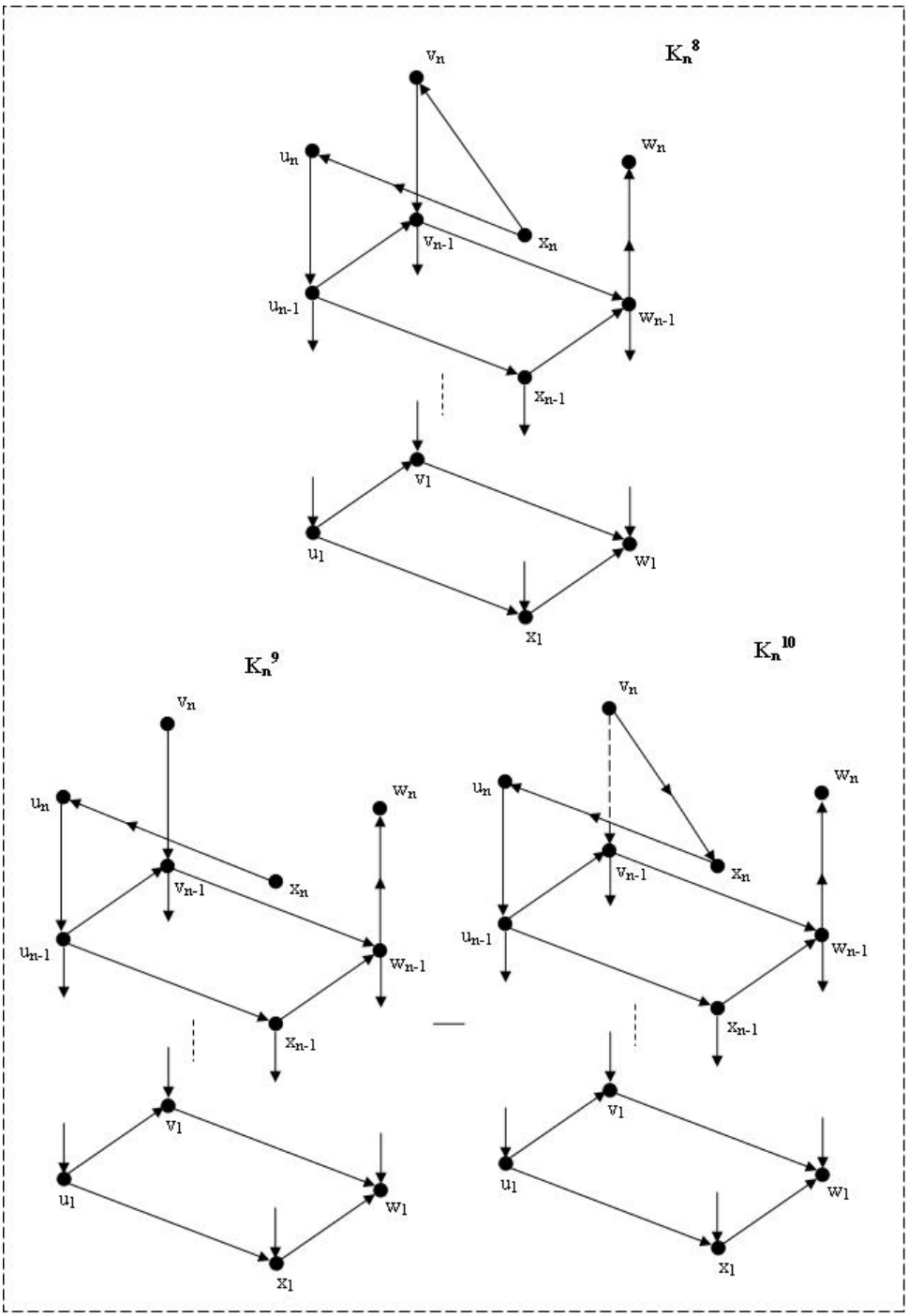

Figure 5.12: Step 10 of Decomposition

Step 11: In this step we reduce H'_n from step 1. We apply Rule 5 on the edge (v_n, v_{n-1}). We get the two resultant graphs K'_n and P'_n as in Figure 5.13. The edge (u_n, v_n) becomes redundant in the graph P'_n and can be discarded according to Rule 2.

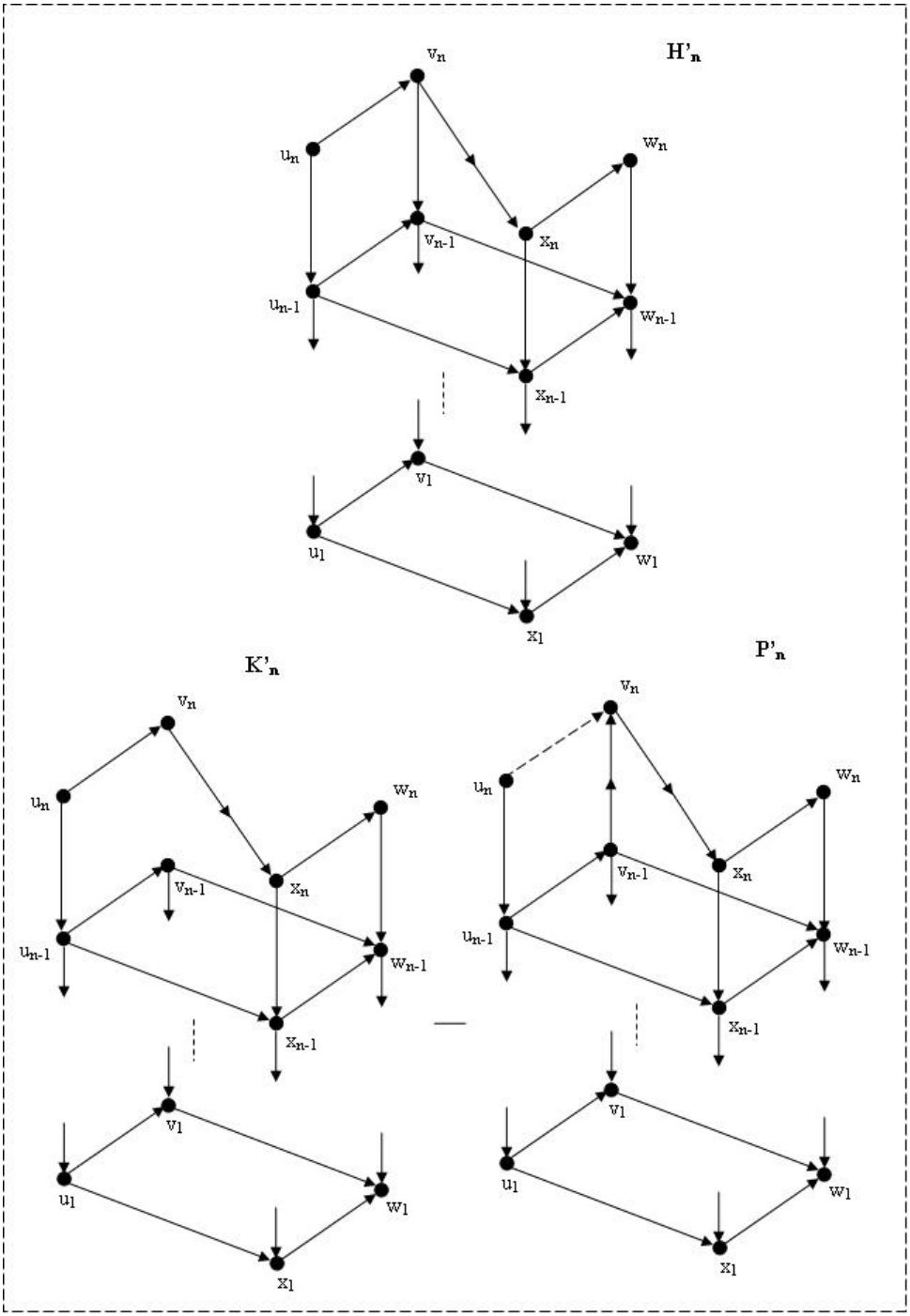

Figure 5.13: Step 11 of Decomposition

Step 12: In this step, we take the graph P'_n and apply Rule 5 on the edge (w_n, w_{n-1}). The resultant graphs are R'_n and S'_n (Figure 5.14).

The edge (x_n, w_n) becomes redundant in the graph S'_n and can be discarded. Both these graphs can be reduced to D'_n shown in Figure 5.15.

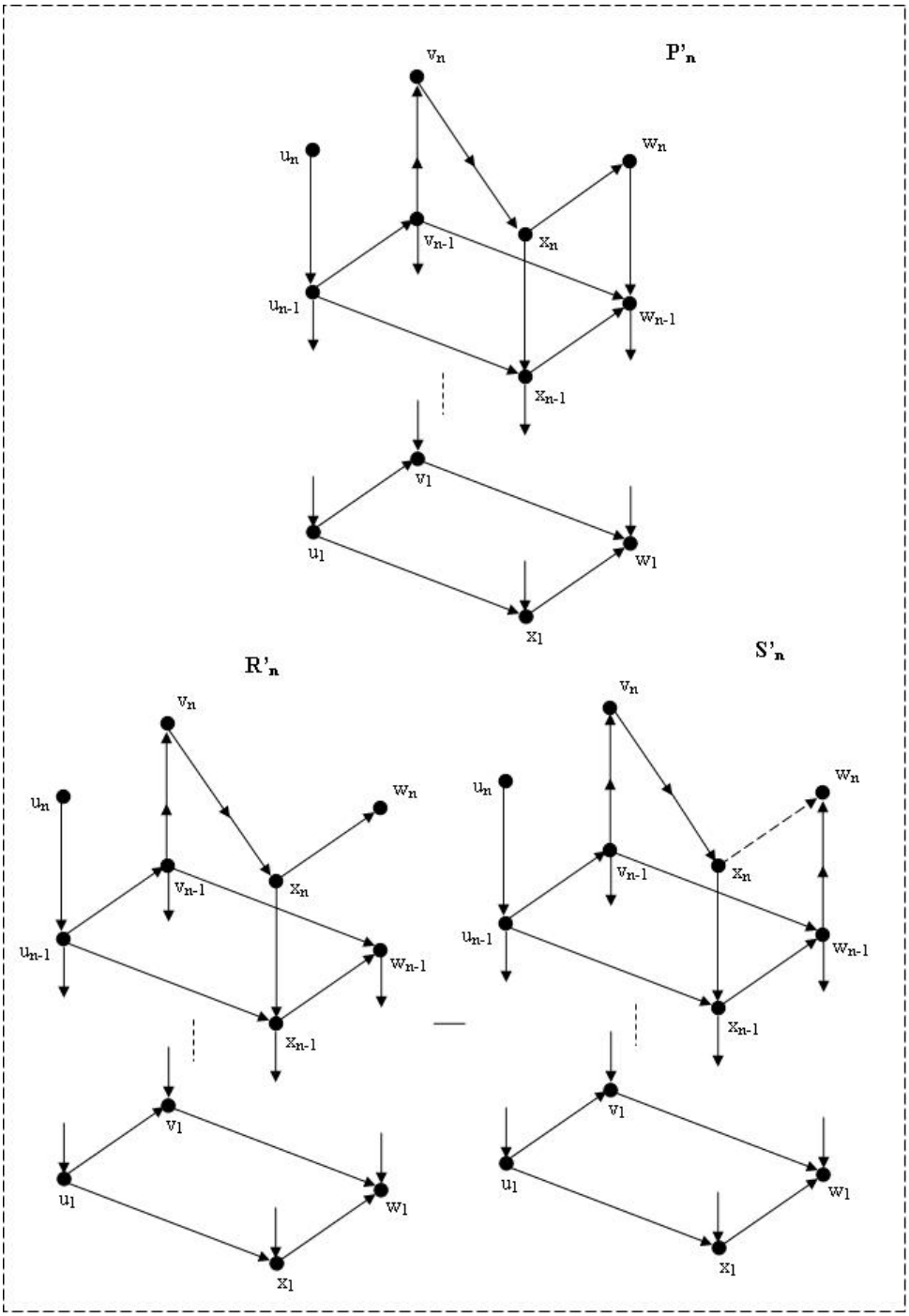

Figure 5.14: Step 12 of Decomposition

D'_n can be derived from R'_n by applying Rule 3 on the edge (u_n, u_{n-1}). On the edge (x_n, w_n), Rule 5 is applied, and to the resultant 2 graphs, Rule 3 and Rule 1 are applied. On applying all these, we get the graph D'_n in each of these cases. Similarly S'_n can be derived from D'_n using Rule 3, Rule 5 and Rule 1.

Next, to decompose D'_n, we first use Rule 2 and again as before, not to remove a redundant edge but to add a redundant edge to the graph. We add the edge (v_{n-1}, x_{n-1}) which is not only redundant, but also is strict, because we want to try to reduce this graph to H'_n (Figure 5.16).

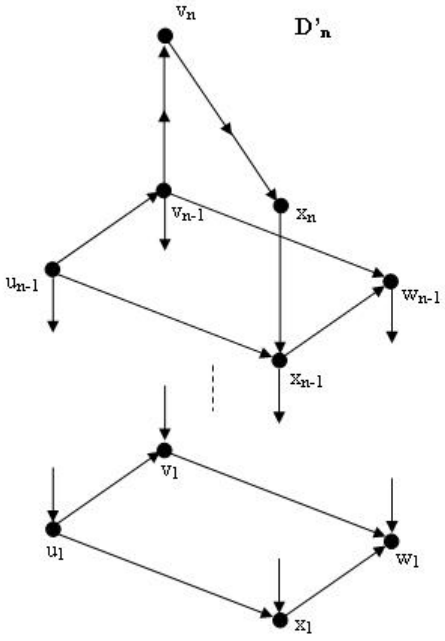

Figure 5.15: The Graph D'_n

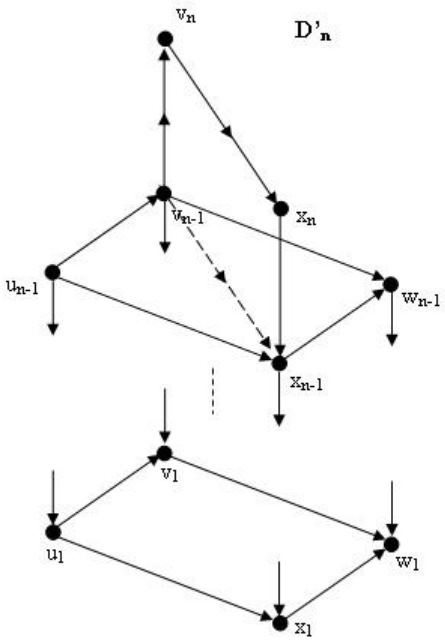

Figure 5.16: The Graph D'_n with the redundant edge

Step 13: In the next step, we take D'_n with the redundant edge and apply Rule 5 on the edge (v_{n-1}, v_n) and we get the two graphs T'_n and W'_n as shown in Figure 5.17.

The graph T'_n can be reduced using Rule 3 on the two edges to get H'_{n-1}. As for the graph W'_n, it can be further reduced as in step 14.

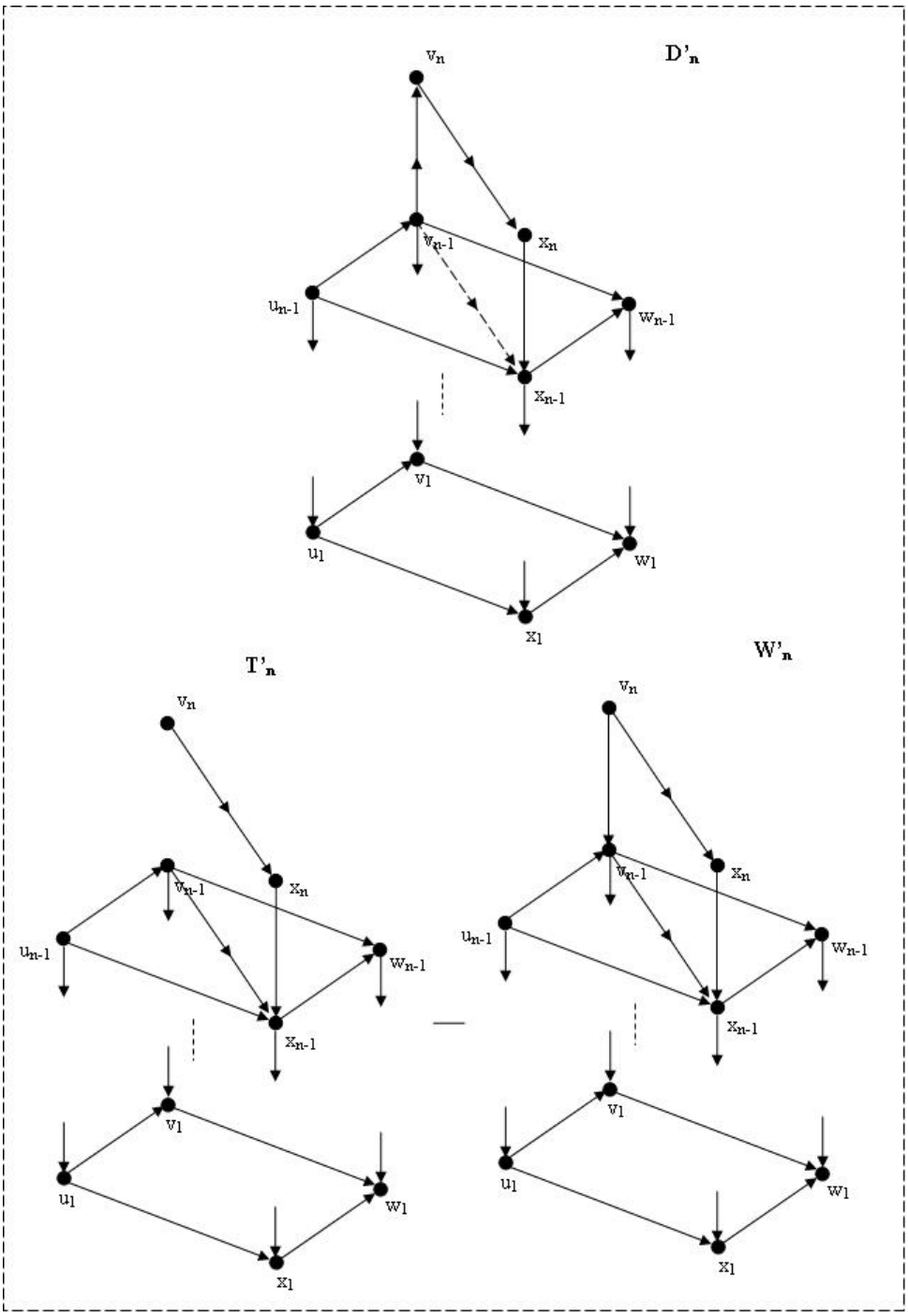

Figure 5.17: Step 13 of Decomposition

Step 14: In this step we take the graph W'_n and apply Rule 5 on the edge (v_n, x_n). We get the two graphs U'_n and V'_n as shown in Figure 5.18.

The edge (x_n, x_{n-1}) becomes redundant in the graph V'_n and can be discarded. Both U'_n and V'_n can be reduced to the graph H'_{n-1} using Rule 3 on the two extra edges. We have reduced the graph P'_n from step 11 to H'_{n-1}. Now we take up the graph K'_n and reduce it.

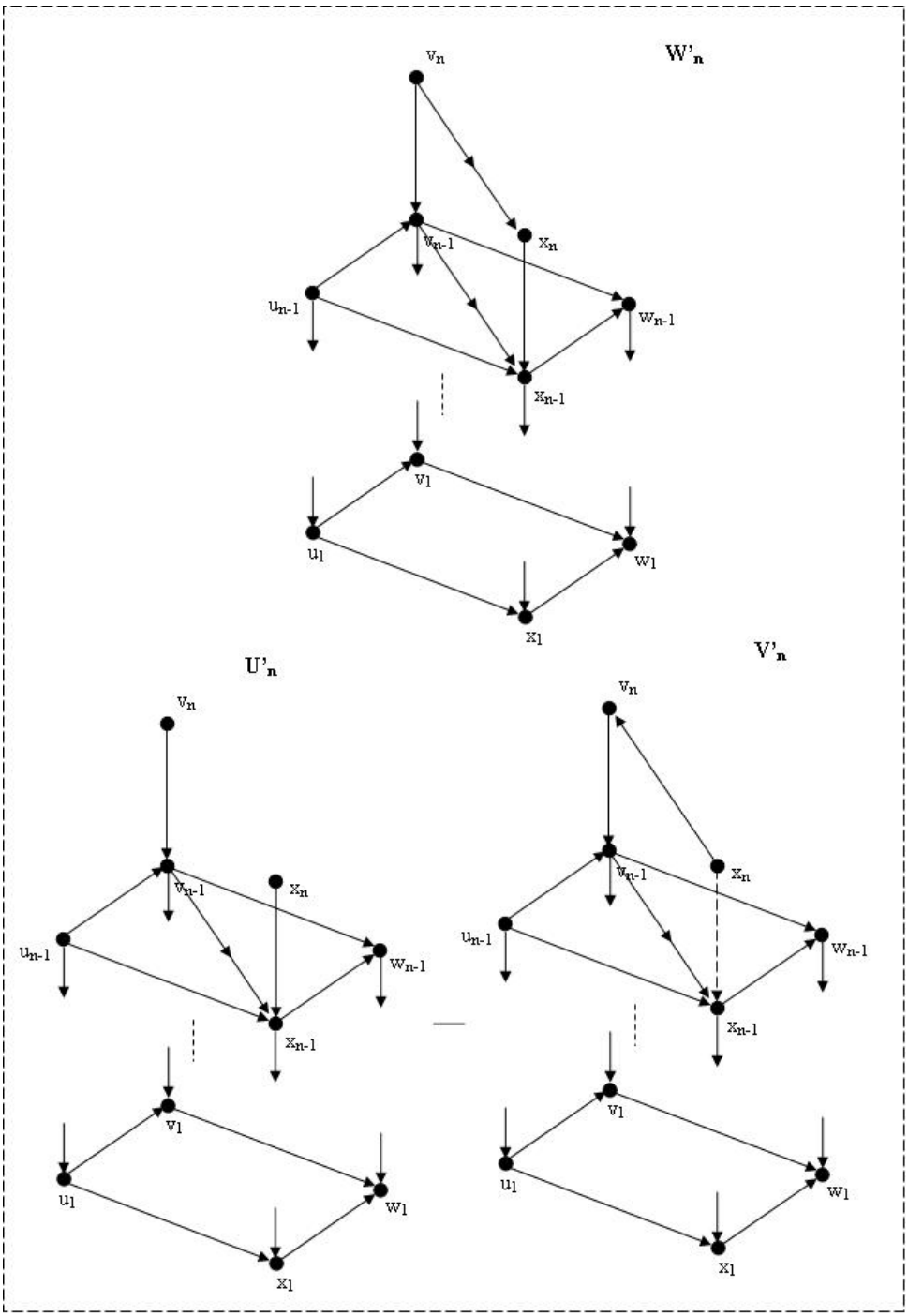

Figure 5.18: Step 14 of Decomposition

Step 15: In this step, we first use Rule 5 on the edge (w_n, w_{n-1}). We get the two graphs $K_n^{'1}$ and $K_n^{'2}$ as shown in Figure 5.19. The edge (x_n, w_n) is redundant in the graph $K_n^{'2}$.

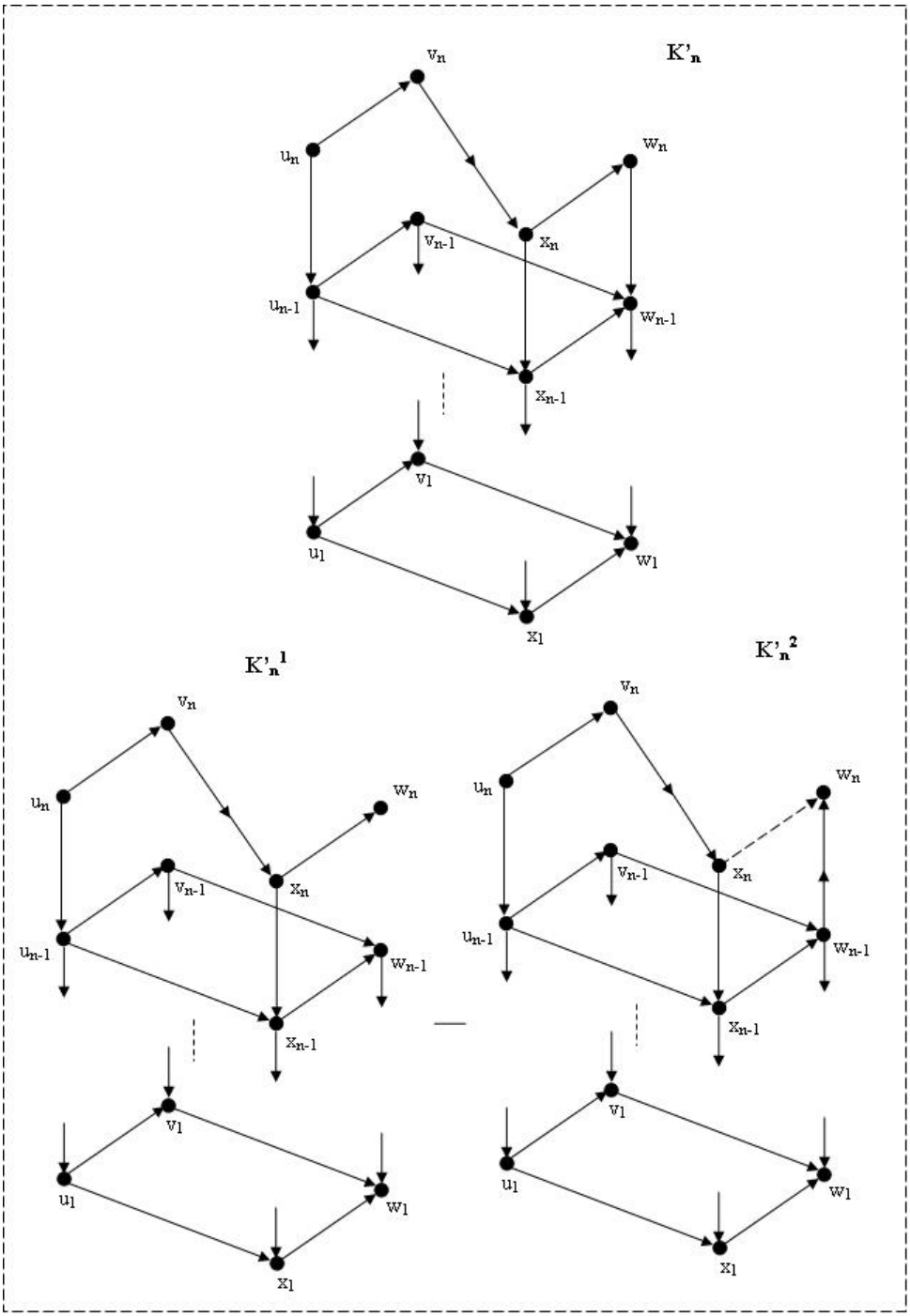

Figure 5.19: Step 15 of Decomposition

Step 16: In the next step, we take the graph $K_n^{'1}$ and apply Rule 5 on the edge (u_n, v_n). We get the 2 graphs $K_n^{'3}$ and $K_n^{'4}$ as shown in Figure 5.20. The graph $K_n^{'3}$ can be reduced to G_{n-1} which will be shown later. Next, we reduce the graph $K_n^{'4}$ further.

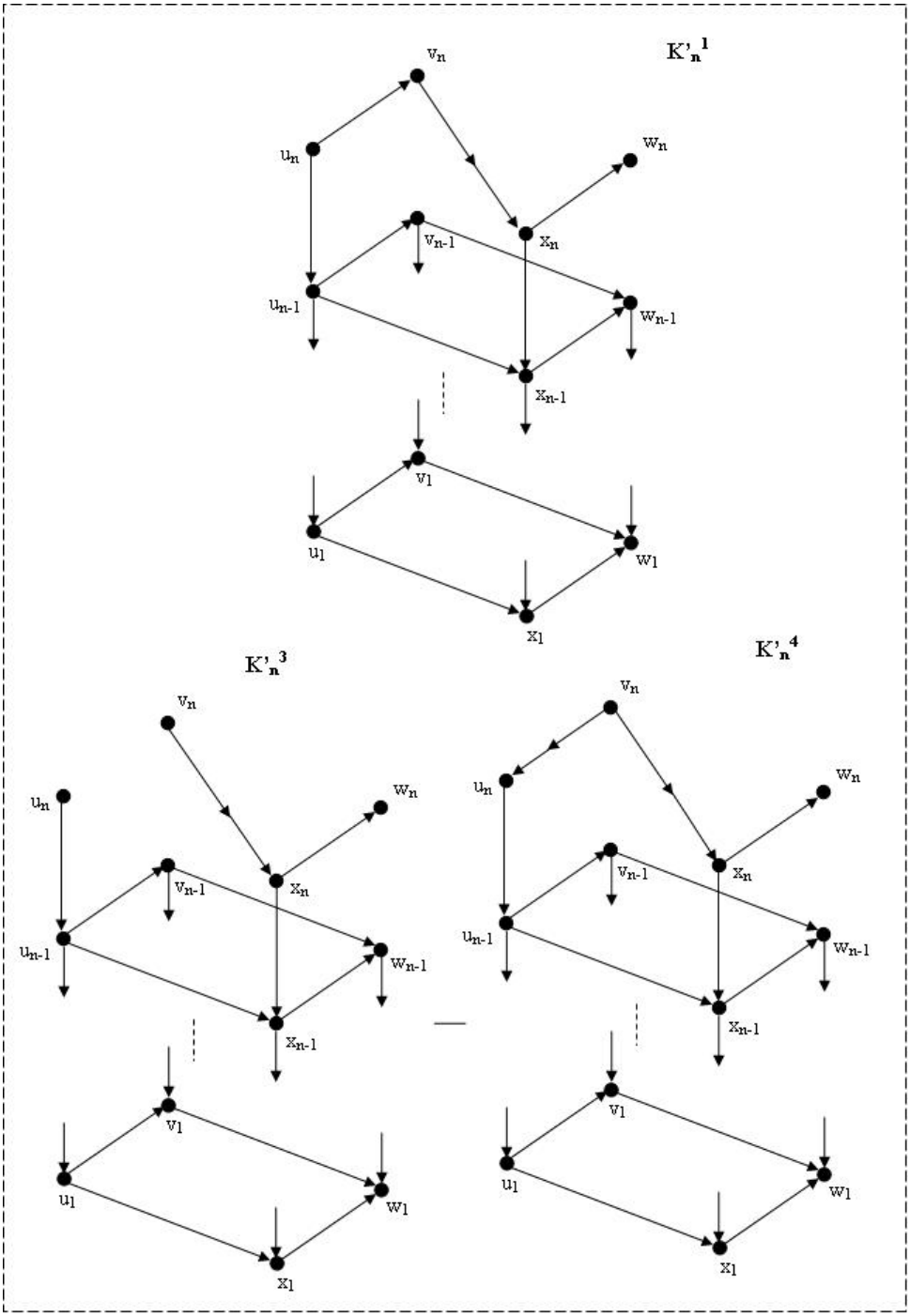

Figure 5.20: Step 16 of Decomposition

Step 17: In this step, we take the graph $K_n^{'4}$, and apply Rule 5 on the edge (v_n, x_n). We get the two graphs $K_n^{'5}$ and $K_n^{'6}$ as shown in Figure 5.21.

The edge (x_n, x_{n-1}) becomes redundant in the graph $K_n^{'6}$. Both the graphs, $K_n^{'5}$ and $K_n^{'6}$ can be reduced to G_{n-1} which will be shown later. We have reduced the graph $K_n^{'1}$ from step 15 and now we have to reduce the graph $K_n^{'2}$.

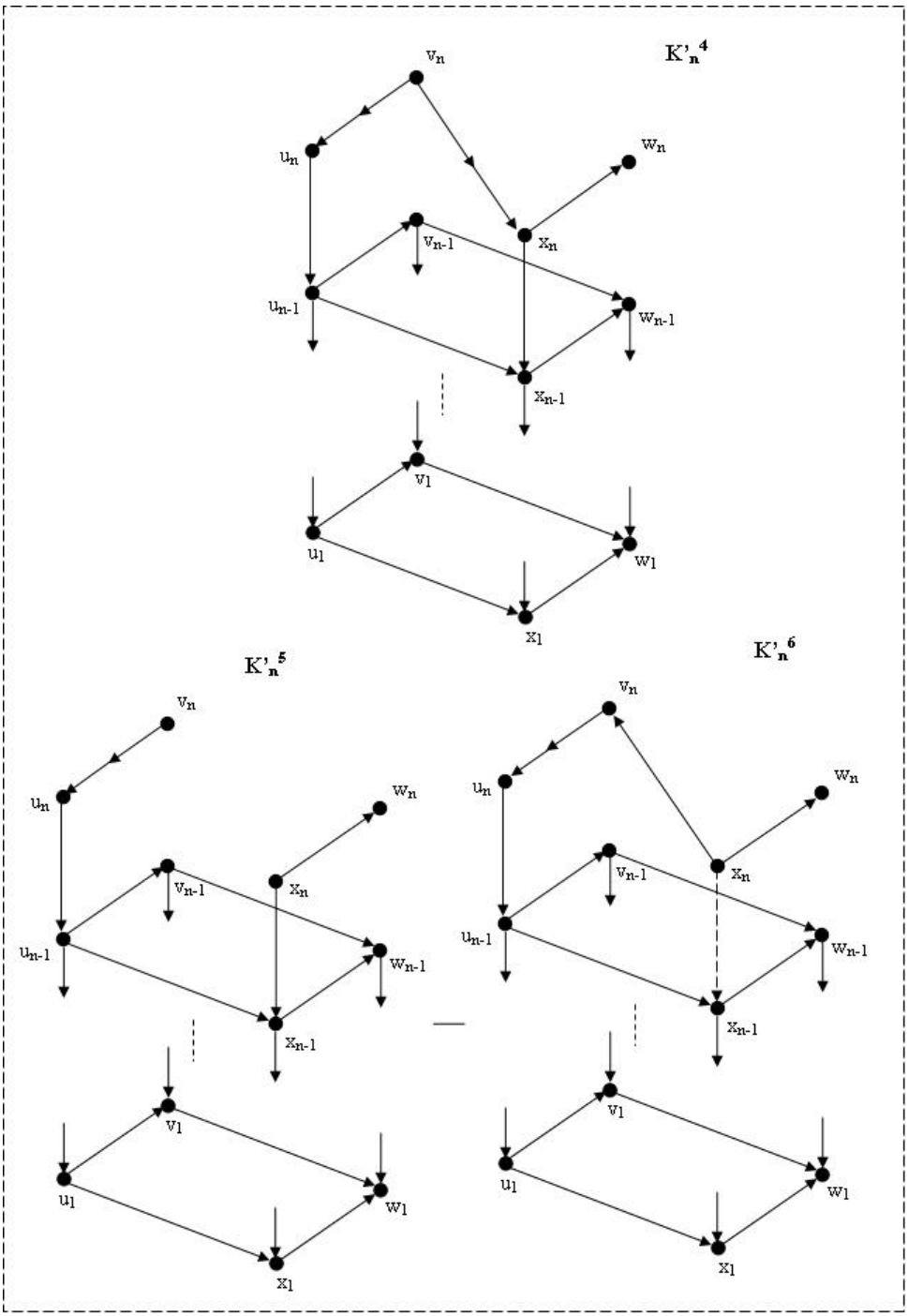

Figure 5.21: Step 17 of Decomposition

Step 18: In this step, we take the graph $K_n^{'2}$ and apply Rule 5 on the edge (u_n, v_n) and we get the 2 graphs $K_n^{'7}$ and $K_n^{'8}$ (Figure 5.22). The graph $K_n^{'7}$ can be reduced to G_{n-1} which will be shown later. The graph $K_n^{'8}$ has to be further simplified.

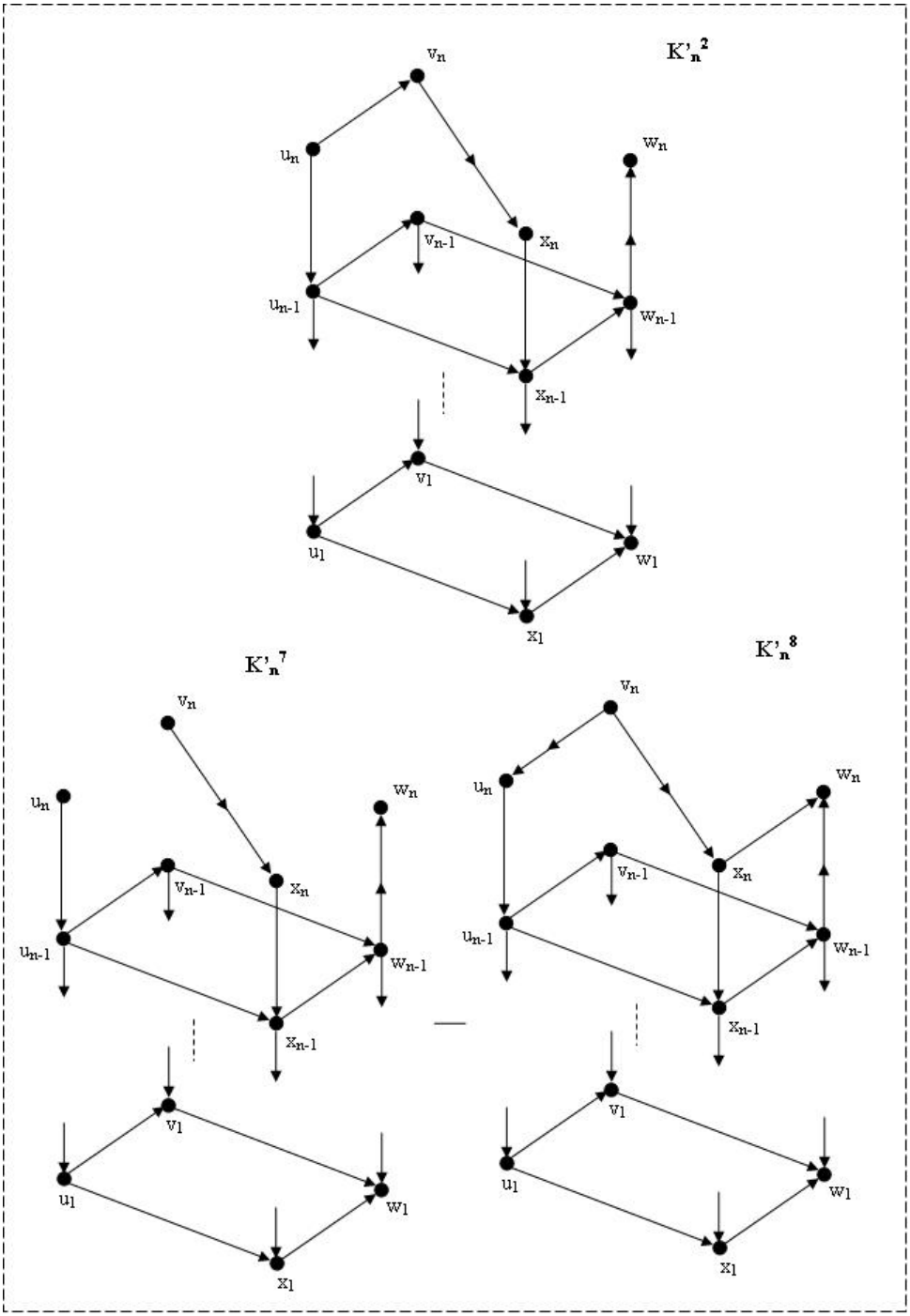

Figure 5.22: Step 18 of Decomposition

Step 19: In this step, the graph $K_n^{'8}$ and apply Rule 5 on the edge (v_n, x_n) and we get the two graphs $(K_n^{'9})$ and $(K_n^{'10})$ as shown in the Figure 5.23.

Both $K_n^{'9}$ and $K_n^{'10}$ can be reduced to G_{n-1} which will be shown later. We have now reduced H_n' from step 1 into 6 instances of G_{n-1} and 6 instances of H_{n-1}. We have reduced the graph G_n now to G_{n-1} and H_{n-1}.

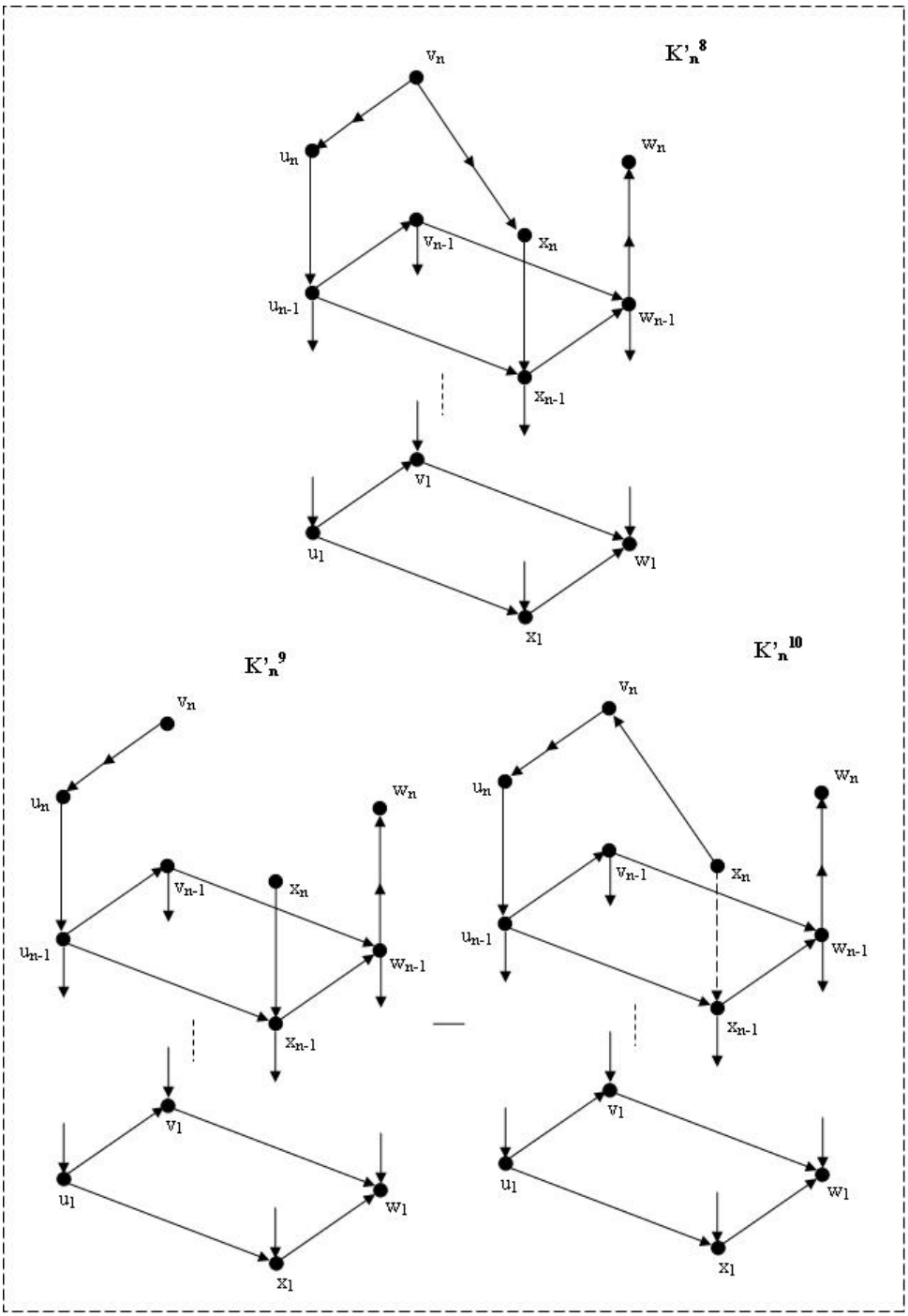

Figure 5.23: Step 19 of Decomposition

5.2 Derivation of a recurrence for the generating function

Now, we proceed to derive the recurrence for the generating function of G_n using the digraph rules. Let the multi-variable generating function of the graph G_n be

$$G(n, u_1, v_1, w_1, x_1, ..., u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n).$$

Since we only manipulate the last 8 vertices of the graph, we are not concerned with the remaining vertices. So we shall compute a 8-variable generating function of G_n given by

$$G(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n),$$

where we use the variable q to represent all the intermediate vertices
$u_i, v_i, w_i, x_i,\ i \in [1..n-2]$ of the G_n. Similarly, let the 8 variable generating function of H_n be

$$H(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

and H'_n be

$$H'(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n).$$

Based on Figure 5.1 and Rule 4 from Section 4.2,

$$G(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) = H(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

$$+H'(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n). \tag{5.1}$$

Figure 5.2 gives us the generating function of the graph H_n with respect to the graphs K_n and P_n. If the 8 variable generating function of the graphs K_n and P_n are

$$K(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

and

$$P(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

respectively, then using Rule 5, we get

$$H(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = K(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-P(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.2}$$

Again from Figure 5.3, using Rule 5, we get the generating function of P_n as

$$P(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = R(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-S(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n), \tag{5.3}$$

where

$$R(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

and

$$S(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

are the 8 variable generating functions of R_n and S_n respectively. We now derive R_n and S_n from the graph D_n shown in Figure 5.4. First, we derive R_n from D_n. R_n can be further broken down using Rule 5 as shown in Figure 5.24.

Both R_n^1 and R_n^2 can be derived from D_n using Rule 3 and Rule 1. We get the generating function

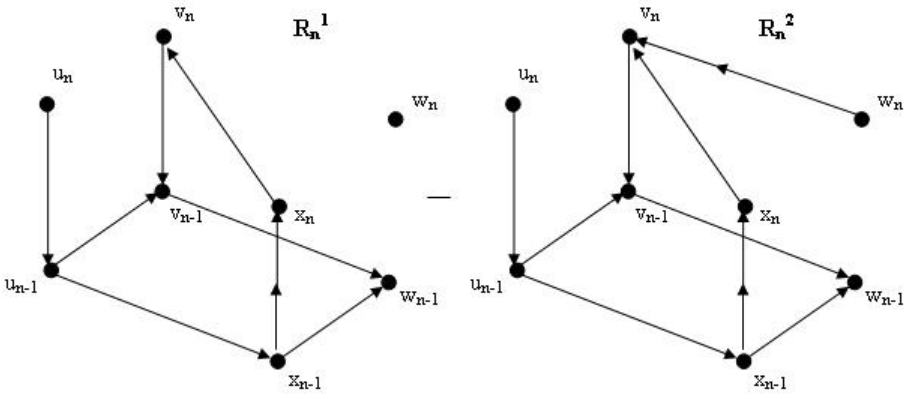

Figure 5.24: Breaking down R_n

of R_n^1 from D_n using Rule 3 on the edge (u_n, u_{n-1}) and Rule 1 on the vertex w_n as

$$R^1(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) =$$

$$D(n, q, (u_n * u_{n-1}), v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) * \frac{(u_n)^0}{(1-u_n)} * \frac{1}{(1-w_n)}. \tag{5.4}$$

Now we get the generating function of R_n^2 from in terms of D_n by using Rule 3 on the edges (u_n, u_{n-1}) and (w_n, v_n) as

$$R^2(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) =$$

$$D(n, q, (u_n * u_{n-1}), v_{n-1}, w_{n-1}, x_{n-1}, u_n, (v_n * w_n), w_n, x_n) * \frac{(u_n)^0}{(1-u_n)} * \frac{(w_n)^1}{(1-w_n)}. \tag{5.5}$$

Here $R^1(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$ is the 8 variable generating function of R_n^1 and $R^2(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$ is the 8 variable generating function of R_n^2. From Figure 5.24, using Rule 5, we get

$$R(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) = R^1(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

$$-R^2(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n). \tag{5.6}$$

Using the Equations 5.6, 5.4 and 5.5 we get

$$R(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) = \frac{D(n, q, u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D(n, q, u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n w_n, w_n, x_n) w_n}{(1-u_n)(1-w_n)}. \tag{5.7}$$

The graph S_n from Figure 5.3 can be further broken down using Rule 5 as shown in Figure 5.25. We can see that R_n^1 and S_n^1 are the same graphs and hence have the same generating function as defined by Equation 5.4. The 8 variable generating function of S_n^2,

$$S_2(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n),$$

is given by

$$S^2(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) =$$

$$D(n, q, (u_n * u_{n-1}), v_{n-1}, (w_n * w_{n-1}), x_{n-1}, u_n, v_n, w_n, x_n) * \frac{(u_n)^0}{(1-u_n)} * \frac{(w_n)^0}{(1-w_n)}. \tag{5.8}$$

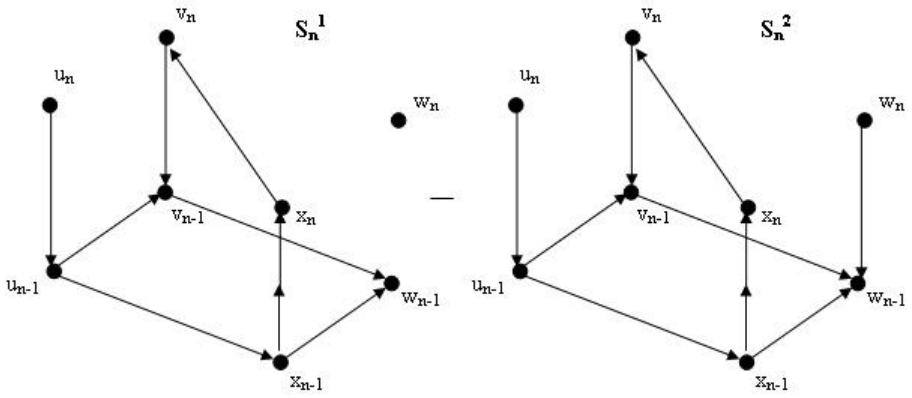

Figure 5.25: Breaking down S_n

Thus the generating function of S_n from Equations 5.4 ,5.8 is given by

$$S(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$= \frac{D(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}. \tag{5.9}$$

From Eqs. 5.3, 5.7, 5.9 we get

$$P(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$= \frac{D(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_nw_n,w_n,x_n)w_n}{(1-u_n)(1-w_n)}. \tag{5.10}$$

Figure 5.6 shows the decomposition of D_n into T_n and W_n. Using Rule 5, we get

$$D(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = T(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-W(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.11}$$

T_n can be derived from H_{n-1} using Rule 3 as follows

$$T(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$H(n-1,q,q,q,q,q,u_{n-1},(x_n*v_n*v_{n-1}),w_{n-1},x_{n-1})*\frac{(x_n*v_n)^0}{(1-(x_n*v_n))}*\frac{(x_n)^0}{(1-x_n)}. \tag{5.12}$$

From Figure 5.7, we get the generating function of W_n in terms of U_n and V_n using Rule 5 as

$$W(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = U(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-V(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.13}$$

We can get the generating function of U_n and V_n in terms of H_{n-1} using Rule 3 as

$$U(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$H(n-1,q,q,q,q,q,u_{n-1},(v_n*v_{n-1}),w_{n-1},(x_n*x_{n-1}))*\frac{(v_n)^0}{(1-v_n)}*\frac{(x_n)^0}{(1-x_n)}, \tag{5.14}$$

and

$$V(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$H(n-1,q,q,q,q,q,u_{n-1},v_{n-1},w_{n-1},(v_n*x_n*x_{n-1}))*\frac{(x_n*v_n)^0}{(1-(x_n*v_n))}*\frac{(v_n)^1}{(1-v_n)}. \tag{5.15}$$

From Eqs. 5.11, 5.12, 5.13, 5.14 and 5.15, we get

$$D(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = \frac{H(n-1,q,q,q,q,q,u_{n-1},x_nv_nv_{n-1},w_{n-1},x_{n-1})}{(1-x_nv_n)(1-x_n)}$$

$$-\frac{H(n-1,q,q,q,q,q,u_{n-1},v_nv_{n-1},w_{n-1},x_nx_{n-1})}{(1-v_n)(1-x_n)}$$

$$+\frac{H(n-1,q,q,q,q,q,u_{n-1},v_{n-1},w_{n-1},v_nx_nx_{n-1})v_n}{(1-x_nv_n)(1-v_n)}. \tag{5.16}$$

We have got the generating function of P_n from Figure 5.2 in terms of H_{n-1}. Now we proceed to get the generating function of K_n. From Figures 5.8, 5.9, 5.10, 5.11 and 5.12 we get the generating function of K_n in terms of K_n^3, K_n^5, K_n^6, K_n^7, K_n^9 and K_n^{10} using Rule 5 as

$$K(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = K^3(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-K^5(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) + K^6(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-K^7(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) + K^9(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

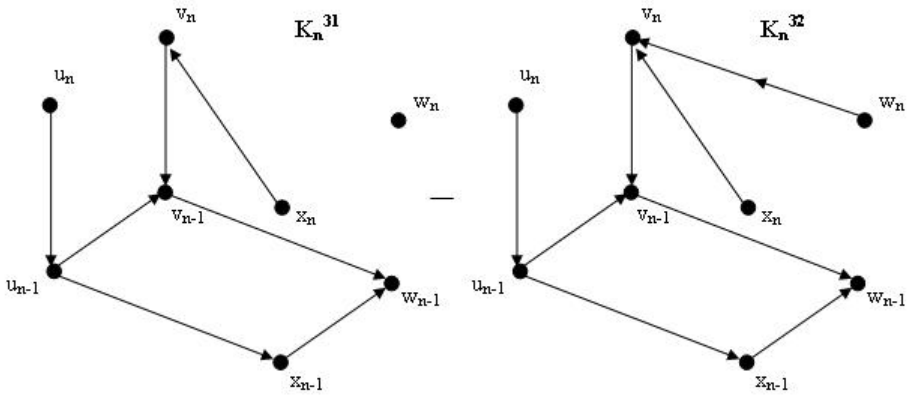

Figure 5.26: Breaking down K_n^3

$$-K^{10}(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n). \tag{5.17}$$

We now get the generating functions of the graphs K_n^3, K_n^5, K_n^6, K_n^7, K_n^9 and K_n^{10} from G_{n-1} as follows. The graph K_n^3 can be further broken down as shown in Figure 5.26. We can get the generating function of K_n^{31} and K_n^{32} from Figure 5.26 using Rule 3 and Rule 1 and we can derive the generating function of K_n^3 from that using Rule 5. We get

$$\begin{aligned} &K^3(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) \\ &= \frac{G(n-1, q, q, q, q, q, u_n u_{n-1}, x_n v_n v_{n-1}, w_{n-1}, x_{n-1})}{(1-u_n)(1-x_n v_n)(1-x_n)(1-w_n)} \\ &- \frac{G(n-1, q, q, q, q, q, u_n u_{n-1}, w_n x_n v_n v_{n-1}, w_{n-1}, x_{n-1}) w_n}{(1-u_n)(1-w_n x_n v_n)(1-x_n)(1-w_n)}. \end{aligned} \tag{5.18}$$

K_n^5 can be further reduced using the inclusion exclusion principle as shown in Figure 5.27 and the generating function of K_n^5 can thus be got as

$$\begin{aligned} &K^5(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) \\ &= \frac{G(n-1, q, q, q, q, q, x_n u_n u_{n-1}, v_n v_{n-1}, w_{n-1}, x_{n-1}) x_n}{(1-x_n u_n)(1-v_n)(1-x_n)(1-w_n)} \\ &- \frac{G(n-1, q, q, q, q, q, x_n u_n u_{n-1}, w_n v_n v_{n-1}, w_{n-1}, x_{n-1}) x_n w_n}{(1-x_n u_n)(1-w_n v_n)(1-x_n)(1-w_n)}. \end{aligned} \tag{5.19}$$

K_n^6 can be further reduced using Rule 5 as shown in Figure 5.28 and the generating function of K_n^6 can thus be got as

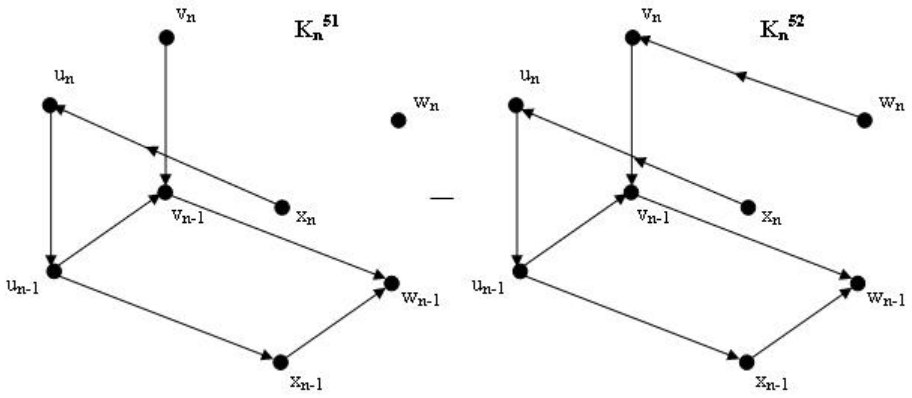

Figure 5.27: Breaking down K_n^5

$$K^6(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

$$= \frac{G(n-1, q, q, q, q, q, v_n x_n u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}) x_n v_n^2}{(1 - v_n x_n u_n)(1 - v_n)(1 - v_n x_n)(1 - w_n)}$$

$$- \frac{G(n-1, q, q, q, q, q, w_n v_n x_n u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}) x_n v_n^2 w_n^3}{(1 - w_n v_n x_n u_n)(1 - w_n v_n)(1 - w_n v_n x_n)(1 - w_n)}. \tag{5.20}$$

K_n^7 can be further reduced using Rule 5 as shown in Figure 5.29 and the generating function of K_n^7 can thus be got as

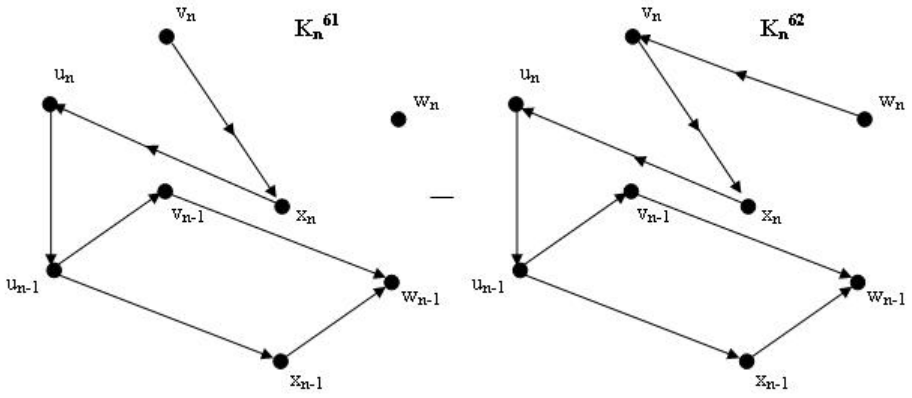

Figure 5.28: Breaking down K_n^6

$$K^7(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

$$= \frac{G(n-1, q, q, q, q, q, u_n u_{n-1}, x_n v_n v_{n-1}, w_{n-1}, x_{n-1})}{(1 - u_n)(1 - x_n v_n)(1 - x_n)(1 - w_n)}$$

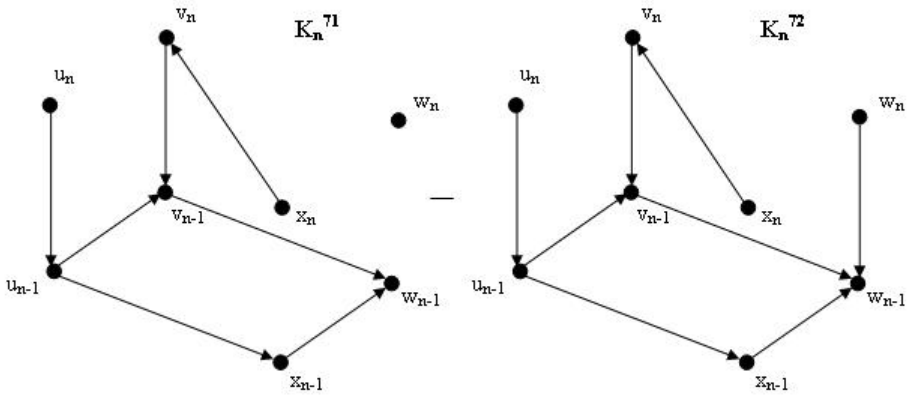

Figure 5.29: Breaking down K_n^7

$$-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},x_nv_nv_{n-1},w_nw_{n-1},x_{n-1})}{(1-u_n)(1-x_nv_n)(1-x_n)(1-w_n)}. \tag{5.21}$$

K_n^9 can be further reduced using Rule 5 as shown in Figure 5.30 and the generating function of K_n^9 can thus be got as

$$\begin{aligned} &K^9(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) \\ &= \frac{G(n-1,q,q,q,q,q,x_nu_nu_{n-1},v_nv_{n-1},w_{n-1},x_{n-1})x_n}{(1-x_nu_n)(1-v_n)(1-x_n)(1-w_n)} \\ &-\frac{G(n-1,q,q,q,q,q,x_nu_nu_{n-1},v_nv_{n-1},w_nw_{n-1},x_{n-1})x_n}{(1-x_nu_n)(1-v_n)(1-x_n)(1-w_n)}. \end{aligned} \tag{5.22}$$

K_n^{10} can be further reduced using Rule 5 as shown in Figure 5.31 and the generating function of K_n^{10} can thus be got as

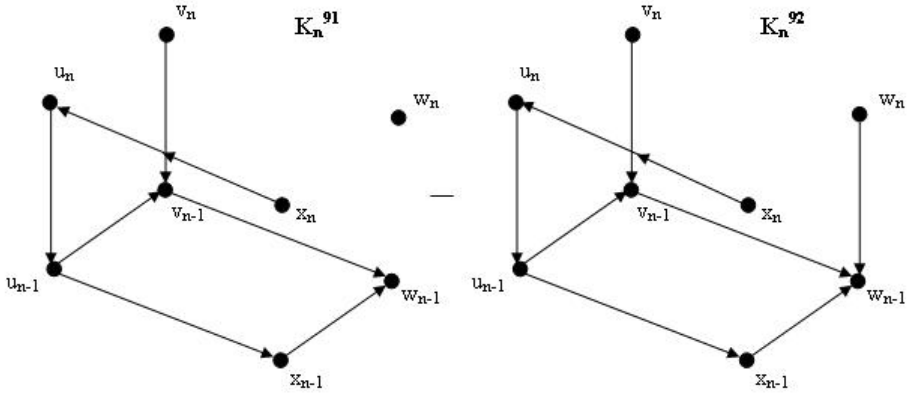

Figure 5.30: Breaking down K_n^9

$$K^{10}(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$$

$$= \frac{G(n-1, q, q, q, q, q, v_n x_n u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}) v_n^2 x_n}{(1 - v_n x_n u_n)(1 - v_n)(1 - v_n x_n)(1 - w_n)}$$

$$- \frac{G(n-1, q, q, q, q, q, v_n x_n u_n u_{n-1}, v_{n-1}, w_n w_{n-1}, x_{n-1}) x_n v_n^2}{(1 - v_n x_n u_n)(1 - v_n)(1 - v_n x_n)(1 - w_n)}. \tag{5.23}$$

From Equations 5.17, 5.18, 5.19, 5.20, 5.21, 5.22 and 5.23 we get the generating function of K_n in terms of G_{n-1} as

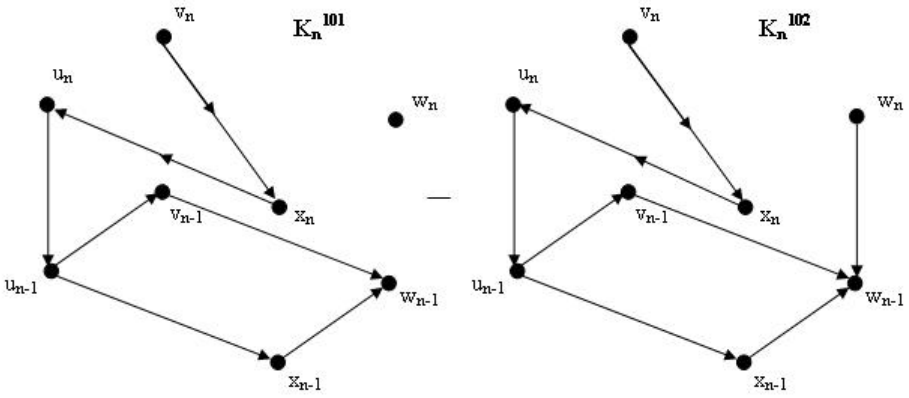

Figure 5.31: Breaking down K_n^{10}

$$K(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) =$$

$$- \frac{G(n-1, q, q, q, q, q, u_n u_{n-1}, w_n x_n v_n v_{n-1}, w_{n-1}, x_{n-1}) w_n}{(1 - u_n)(1 - w_n x_n v_n)(1 - x_n)(1 - w_n)}$$

$$+ \frac{G(n-1, q, q, q, q, q, x_n u_n u_{n-1}, w_n v_n v_{n-1}, w_{n-1}, x_{n-1}) x_n w_n}{(1 - x_n u_n)(1 - w_n v_n)(1 - x_n)(1 - w_n)}$$

$$- \frac{G(n-1, q, q, q, q, q, w_n v_n x_n u_n u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}) x_n v_n^2 w_n^3}{(1 - w_n v_n x_n u_n)(1 - w_n v_n)(1 - w_n v_n x_n)(1 - w_n)}$$

$$+ \frac{G(n-1, q, q, q, q, q, u_n u_{n-1}, x_n v_n v_{n-1}, w_n w_{n-1}, x_{n-1})}{(1 - u_n)(1 - x_n v_n)(1 - x_n)(1 - w_n)}$$

$$- \frac{G(n-1, q, q, q, q, q, x_n u_n u_{n-1}, v_n v_{n-1}, w_n w_{n-1}, x_{n-1}) x_n}{(1 - x_n u_n)(1 - v_n)(1 - x_n)(1 - w_n)}$$

$$+ \frac{G(n-1, q, q, q, q, q, v_n x_n u_n u_{n-1}, v_{n-1}, w_n w_{n-1}, x_{n-1}) x_n v_n^2}{(1 - v_n x_n u_n)(1 - v_n)(1 - v_n x_n)(1 - w_n)}. \tag{5.24}$$

Now, we can consolidate Equations 5.2, 5.10, 5.16, 5.24 and get the recurrence for H_n as

$$H(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$(-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},w_nx_nv_nv_{n-1},w_{n-1},x_{n-1})w_n}{(1-u_n)(1-w_nx_nv_n)(1-x_n)(1-w_n)}$$

$$+\frac{G(n-1,q,q,q,q,q,x_nu_nu_{n-1},w_nv_nv_{n-1},w_{n-1},x_{n-1})x_nw_n}{(1-x_nu_n)(1-w_nv_n)(1-x_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,w_nv_nx_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})x_nv_n^2w_n^3}{(1-w_nv_nx_nu_n)(1-w_nv_n)(1-w_nv_nx_n)(1-w_n)}$$

$$+\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},x_nv_nv_{n-1},w_nw_{n-1},x_{n-1})}{(1-u_n)(1-x_nv_n)(1-x_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,x_nu_nu_{n-1},v_nv_{n-1},w_nw_{n-1},x_{n-1})x_n}{(1-x_nu_n)(1-v_n)(1-x_n)(1-w_n)})$$

$$-(\frac{D(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_nw_n,w_n,x_n)w_n}{(1-u_n)(1-w_n)}),$$

where

$$D(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$\frac{H(n-1,q,q,q,q,q,u_{n-1},x_nv_nv_{n-1},w_{n-1},x_{n-1})}{(1-x_nv_n)(1-x_n)}$$

$$-\frac{H(n-1,q,q,q,q,q,u_{n-1},v_nv_{n-1},w_{n-1},x_nx_{n-1})}{(1-v_n)(1-x_n)}$$

$$+\frac{H(n-1,q,q,q,q,q,u_{n-1},v_{n-1},w_{n-1},v_nx_nx_{n-1})v_n}{(1-x_nv_n)(1-v_n)}. \tag{5.25}$$

We now find the generating function of H'_n using a similar approach. We briefly describe the method focusing on the final generating function. From Figure 5.13 using Rule 5

$$H'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = K'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-P'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.26}$$

Again from Figure 5.14, and Rule 5, we get the generating function of P'_n as

$$P'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = R'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-S'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.27}$$

Both R'_n and S'_n are derived from D'_n shown in Figure 5.15 as per Figures 5.32 and 5.33. We get

$$R'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = \frac{D'(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,w_nx_n)w_n}{(1-u_n)(1-w_n)} \tag{5.28}$$

and

$$S'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = \frac{D'(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}. \tag{5.29}$$

From Eqs. 5.27, 5.28, 5.29 we get

$$P'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}$$

$$-\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,w_nx_n)w_n}{(1-u_n)(1-w_n)}. \tag{5.30}$$

Figure 5.17 shows the decomposition of D'_n into T'_n and W'_n. Using Rule 5, we get

$$D'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = T'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-W'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.31}$$

From Figure 5.18, we get the generating function of W'_n in terms of U'_n and V'_n using Rule 5 as

$$W'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = U'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

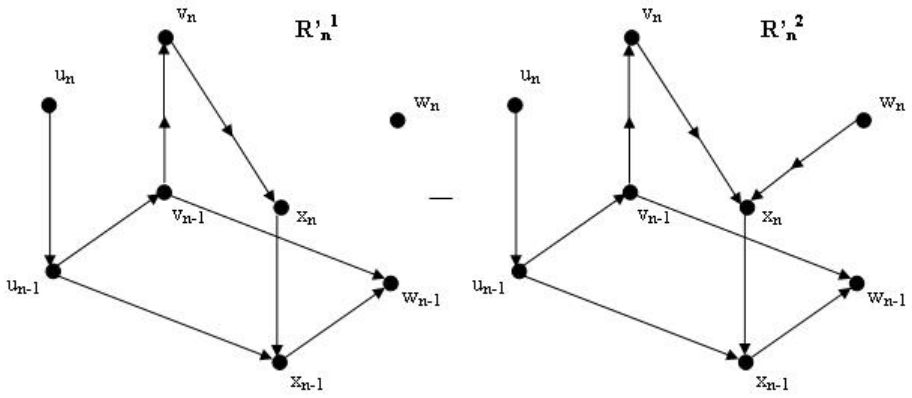

Figure 5.32: Derivation of R'_n from D'_n

$$-V'(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n). \tag{5.32}$$

Using Rule 3 on T'_n, U'_n, V'_n and using Equation 5.31 and 5.32, we get

$$D'(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n) = \frac{H'(n-1, q, q, q, q, q, u_{n-1}, x_n v_n v_{n-1}, w_{n-1}, x_{n-1})}{(1-(x_n * v_n))(1-x_n)}$$

$$-\frac{H'(n-1, q, q, q, q, q, u_{n-1}, v_n v_{n-1}, w_{n-1}, x_n x_{n-1})}{(1-v_n)(1-x_n)}$$

$$+\frac{H'(n-1, q, q, q, q, q, u_{n-1}, v_{n-1}, w_{n-1}, v_n x_n x_{n-1})v_n}{(1-(v_n * x_n))(1-v_n)}. \tag{5.33}$$

From Figures 5.19, 5.20, 5.21, 5.22 and 5.23 we get the generating function of K'_n in terms of K'^3_n,

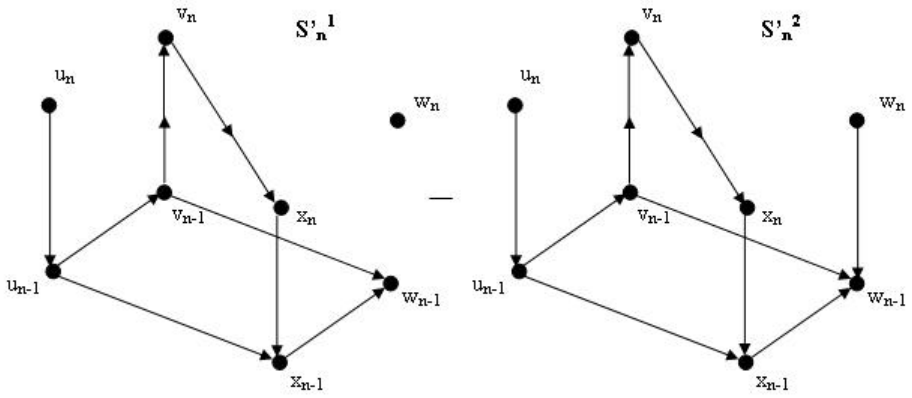

Figure 5.33: Derivation of S'_n from D'_n

$K_n^{'5}$, $K_n^{'6}$, $K_n^{'7}$, $K_n^{'9}$ and $K_n^{'10}$ using Rule 5 as

$$K'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) = K^{'3}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-K^{'5}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) + K^{'6}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-K^{'7}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) + K^{'9}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$-K^{'10}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n). \tag{5.34}$$

We now get the generating functions of the graphs $K_n^{'3}$, $K_n^{'5}$, $K_n^{'6}$, $K_n^{'7}$, $K_n^{'9}$ and $K_n^{'10}$ from G_{n-1} as follows. The graph $K_n^{'3}$ can be further broken down as shown in Figure 5.34. The generating function of K_n^3 is

$$\begin{aligned} &K^{'3}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) \\ &= \frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_{n-1},v_nx_nx_{n-1})v_n}{(1-u_n)(1-v_n)(1-v_nx_n)(1-w_n)} \\ &-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_{n-1},v_nw_nx_nx_{n-1})w_nv_n}{(1-u_n)(1-v_n)(1-v_nw_nx_n)(1-w_n)}. \end{aligned} \tag{5.35}$$

$K_n^{'5}$ can be got from G_{n-1} from Figure 5.35

$$\begin{aligned} &K^{'5}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) \\ &= \frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_{n-1},x_nx_{n-1})v_n}{(1-v_nu_n)(1-v_n)(1-x_n)(1-w_n)} \end{aligned}$$

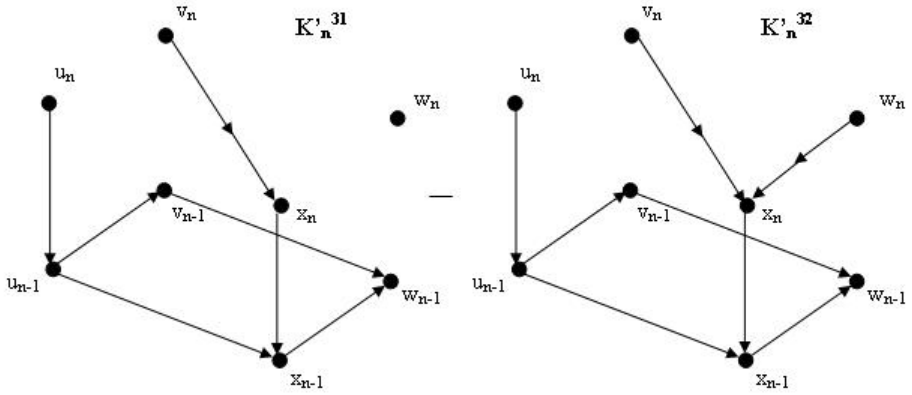

Figure 5.34: Breaking down $K_n^{'3}$

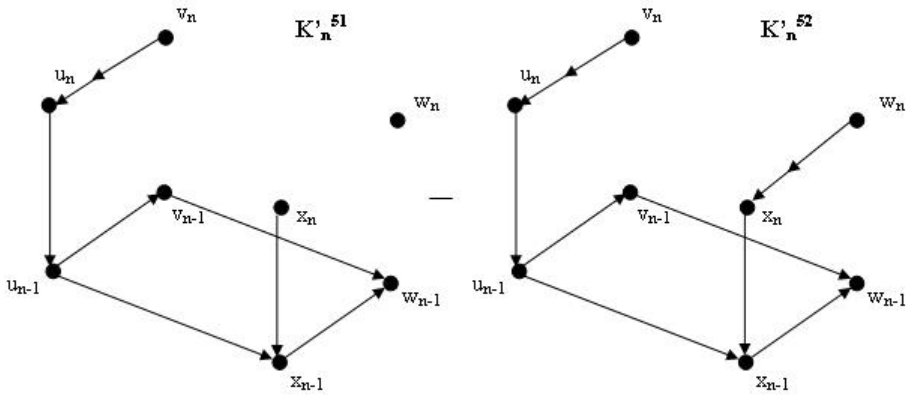

Figure 5.35: Breaking down $K_n^{'5}$

$$-\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_{n-1},w_nx_nx_{n-1})v_nw_n}{(1-v_nu_n)(1-v_n)(1-w_nx_n)(1-w_n)}. \tag{5.36}$$

$K_n^{'6}$ can be got from G_{n-1} from Figure 5.36

$$K^{'6}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$=\frac{G(n-1,q,q,q,q,q,x_nv_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})v_nx_n}{(1-x_nv_nu_n)(1-x_nv_n)(1-x_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,w_nx_nv_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})v_nx_nw_n^2}{(1-w_nx_nv_nu_n)(1-w_nx_nv_n)(1-w_nx_n)(1-w_n)}. \tag{5.37}$$

$K_n^{'7}$ can be got from G_{n-1} from Figure 5.37

$$K^{'7}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

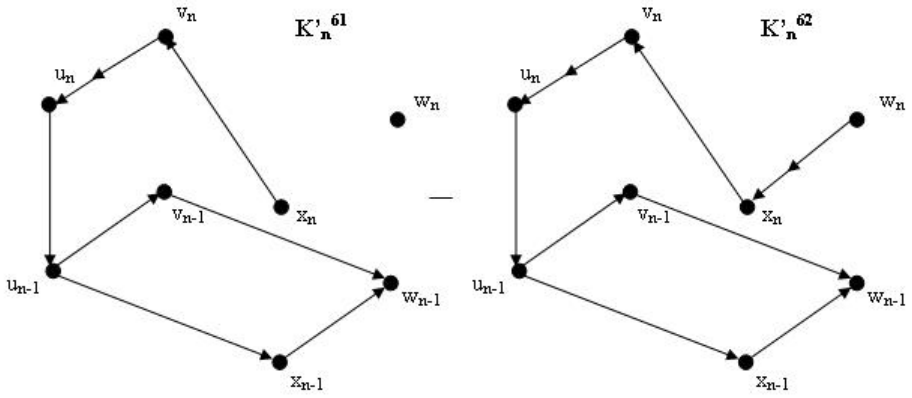

Figure 5.36: Breaking down $K_n^{'6}$

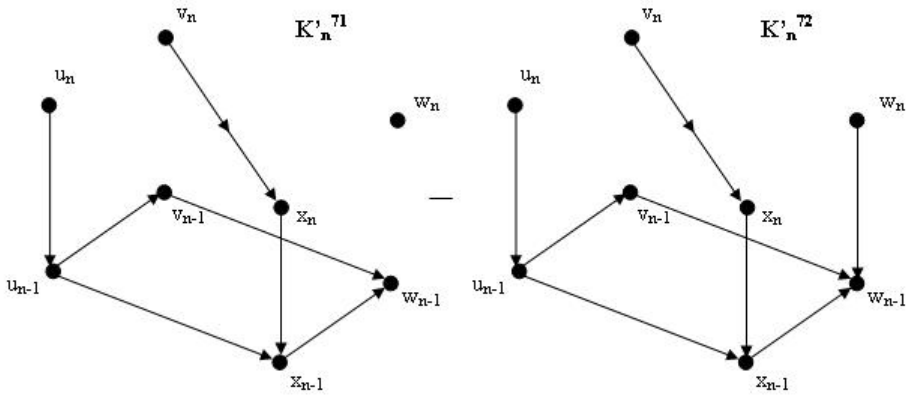

Figure 5.37: Breaking down $K_n^{'7}$

$$= \frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_{n-1},v_nx_nx_{n-1})v_n}{(1-u_n)(1-v_n)(1-v_nx_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},v_nx_nx_{n-1})v_n}{(1-u_n)(1-v_n)(1-v_nx_n)(1-w_n)}. \tag{5.38}$$

$K_n^{'9}$ can be got from G_{n-1} from Figure 5.38

$$K^{'9}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$= \frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_{n-1},x_nx_{n-1})v_n}{(1-v_nu_n)(1-v_n)(1-x_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_nx_{n-1})v_n}{(1-v_nu_n)(1-v_n)(1-x_n)(1-w_n)}. \tag{5.39}$$

$K_n^{'10}$ can be got from G_{n-1} from Figure 5.39

$$K^{'10}(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n)$$

$$= \frac{G(n-1,q,q,q,q,q,x_nv_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})x_nv_n}{(1-x_nv_nu_n)(1-x_nv_n)(1-x_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,x_nv_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1})x_nv_n}{(1-x_nv_nu_n)(1-x_nv_n)(1-x_n)(1-w_n)}. \tag{5.40}$$

From Equations 5.34, 5.35, 5.36, 5.37, 5.38, 5.39 and 5.40 we get the generating function of K'_n in terms of G_{n-1} as

$$K'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

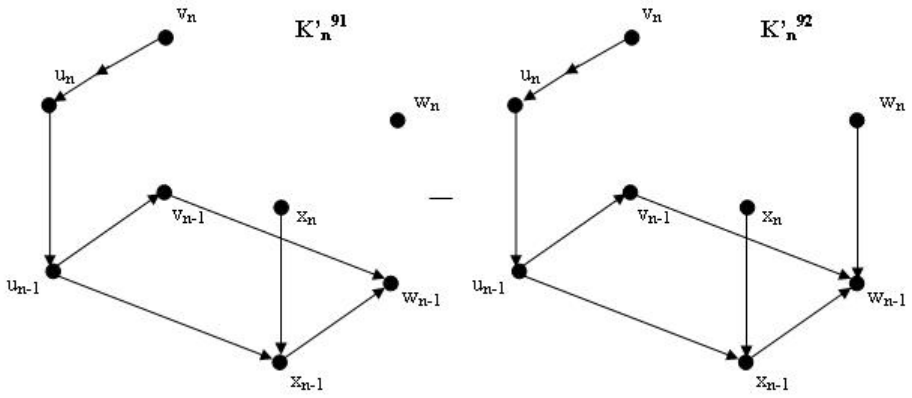

Figure 5.38: Breaking down $K_n^{'9}$

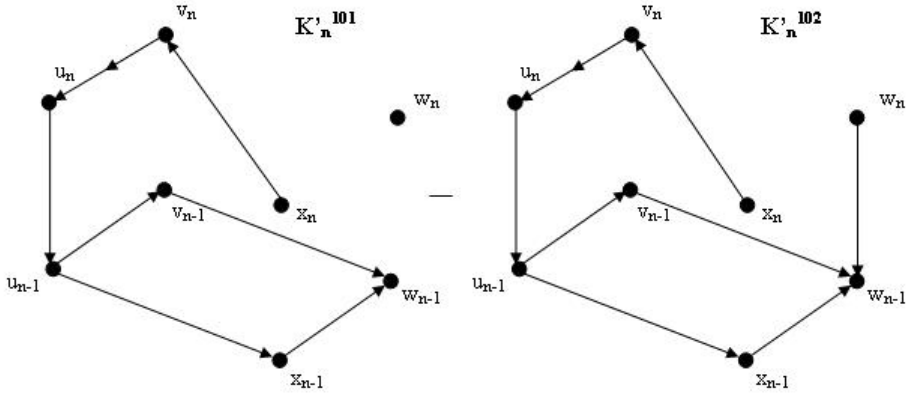

Figure 5.39: Breaking down $K_n^{'10}$

$$-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_{n-1},v_nw_nx_nx_{n-1})w_nv_n}{(1-u_n)(1-v_n)(1-v_nw_nx_n)(1-w_n)}$$

$$+\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_{n-1},w_nx_nx_{n-1})v_nw_n}{(1-v_nu_n)(1-v_n)(1-w_nx_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,w_nx_nv_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})v_nx_nw_n^2}{(1-w_nx_nv_nu_n)(1-w_nx_nv_n)(1-w_nx_n)(1-w_n)}$$

$$+\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},v_nx_nx_{n-1})v_n}{(1-u_n)(1-v_n)(1-v_nx_n)(1-w_n)}$$

$$-\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_nx_{n-1})v_n}{(1-v_nu_n)(1-v_n)(1-x_n)(1-w_n)}$$

$$+\frac{G(n-1,q,q,q,q,q,x_nv_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1})x_nv_n}{(1-x_nv_nu_n)(1-x_nv_n)(1-x_n)(1-w_n)}. \tag{5.41}$$

Now, we can consolidate Equations 5.26, 5.30, 5.33, 5.41 and get the recurrence for H'_n as

$$H'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =$$

$$\begin{aligned}
&(-\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_{n-1},v_nw_nx_nx_{n-1})w_nv_n}{(1-u_n)(1-v_n)(1-v_nw_nx_n)(1-w_n)}\\
&+\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_{n-1},w_nx_nx_{n-1})v_nw_n}{(1-v_nu_n)(1-v_n)(1-w_nx_n)(1-w_n)}\\
&-\frac{G(n-1,q,q,q,q,q,w_nx_nv_nu_nu_{n-1},v_{n-1},w_{n-1},x_{n-1})v_nx_nw_n^2}{(1-w_nx_nv_nu_n)(1-w_nx_nv_n)(1-w_nx_n)(1-w_n)}\\
&+\frac{G(n-1,q,q,q,q,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},v_nx_nx_{n-1})v_n}{(1-u_n)(1-v_n)(1-v_nx_n)(1-w_n)}\\
&-\frac{G(n-1,q,q,q,q,q,v_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_nx_{n-1})v_n}{(1-v_nu_n)(1-v_n)(1-x_n)(1-w_n)}\\
&+\frac{G(n-1,q,q,q,q,q,x_nv_nu_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1})x_nv_n}{(1-x_nv_nu_n)(1-x_nv_n)(1-x_n)(1-w_n)})\\
&\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_nw_{n-1},x_{n-1},u_n,v_n,w_n,x_n)}{(1-u_n)(1-w_n)}\\
&-\frac{D'(n,q,u_nu_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,w_nx_n)w_n}{(1-u_n)(1-w_n)},
\end{aligned}$$

where

$$\begin{aligned}
&D'(n,q,u_{n-1},v_{n-1},w_{n-1},x_{n-1},u_n,v_n,w_n,x_n) =\\
&\frac{H'(n-1,q,q,q,q,q,u_{n-1},x_nv_nv_{n-1},w_{n-1},x_{n-1})}{(1-(x_n*v_n))(1-x_n)}\\
&-\frac{H'(n-1,q,q,q,q,q,u_{n-1},v_nv_{n-1},w_{n-1},x_nx_{n-1})}{(1-v_n)(1-x_n)}\\
&+\frac{H'(n-1,q,q,q,q,q,u_{n-1},v_{n-1},w_{n-1},v_nx_nx_{n-1})v_n}{(1-(v_n*x_n))(1-v_n)}.
\end{aligned} \tag{5.42}$$

We can get the recurrence for G_n using Equations 5.1, 5.25 and 5.42.

5.3 Base Case

Every recurrence must have a base case on which it can be built on. For the $2 \times 2 \times n$ graph the base case is the $2 \times 2 \times 1$ (G_1) graph as shown in Figure 5.40. We split this graph up into two graphs

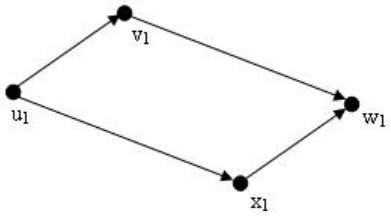

Figure 5.40: The base graph - $2 \times 2 \times 1$

using Rule 4 and get H_1 and H_1' so that we have a base case for both H_n and H_n' as shown in Figure 5.41. We can see from Figure 5.41 that after removing the redundant edges, the two base cases H_1 and H_1' are just two straight line graphs. We can get the generating function for them using Rule 3 repeatedly. The generating functions of H_1 and H_1' are

$$H(1, q, q, q, q, q, u_1, v_1, w_1, x_1) = \frac{1}{(1-u_1)(1-u_1x_1)(1-u_1x_1v_1)(1-u_1x_1v_1w_1)} \tag{5.43}$$

and

$$H'(1, q, q, q, q, q, u_1, v_1, w_1, x_1) = \frac{u_1v_1}{(1-u_1)(1-u_1v_1)(1-u_1v_1x_1)(1-u_1v_1x_1w_1)}. \tag{5.44}$$

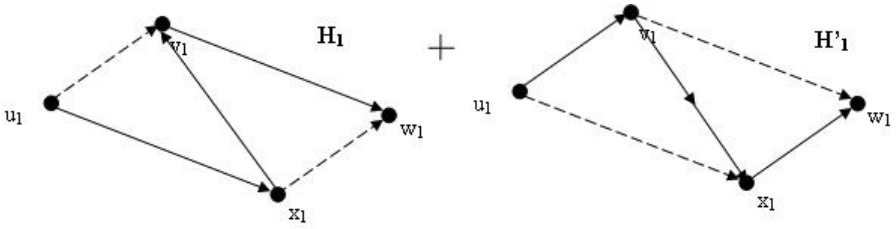

Figure 5.41: H_1 and H_1'

Chapter 6

Automation of the recurrence

6.1 Maple implementation

In Chapter 5, we derived the recurrence for the generating function of $2 \times 2 \times n$ solid partitions represented by the digraph G_n in Figure 4.7. We implemented the recurrence in Maple and the program is shown in Appendix A. Figure 6.1 shows the procedure that implements the first equation in the recurrence given by Equation 5.1 from Section 5.2. Using this program, we can:

1. Compute $G(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$ defined in Section 5.2 by calling:

   > `Gn(n,q,a,b,c,d,u,v,w,x);`.

2. Compute the univariate generating function, $G_{2\times2\times n}(q)$, of $2 \times 2 \times n$ solid partitions by calling:

   > `Gn(n,q,q,q,q,q,q,q,q,q);`.

3. Compute the number of linear extensions of the $2 \times 2 \times n$ poset ($\ell(P_{2\times2\times n})$) using the method

```
Gn := proc(n,q,a,b,c,d,u,v,w,x);
     return (Hn(n,q,a,b,c,d,u,v,w,x)+Hn1(n,q,a,b,c,d,u,v,w,x))
```

Figure 6.1: The first procedure from the program in Appendix A

Table 6.1: Computing the univariate generating function of $2 \times 2 \times n$ solid partitions using the program in Appendix A

n	Time (in Sec.)	Memory (in MB)
1	0	0
2	0	0
3	1	5
4	5	54
5	145	795

described in Section 3.3 by calling:

```
> subs(q=1,simplify(Gn(n,q,q,q,q,q,q,q,q,q)*product('q^i','i'=1..4*n)));,
```

where the Maple function `product('q^i','i'=1..4*n))` gives $\prod_{i=1}^{4n}(1-q^i)$. By multiplying this with the generating function, we get the W-polynomial mentioned in Section 3.2.1. The function `simplify(Q)` simplifies a given expression `Q`. This is required so that W-polynomial is reduced to a polynomial and we can substitute $q = 1$ to get $\ell(P_{2\times 2\times n})$. The Maple function `subs(a=b,Q)` substitutes in `Q`, for all values of a with b.

We ran our program on a machine running Maple 11 with the following configuration:

- Processor: 2.6 GHz AMD Athelon 64x2 dual core processor
- Local memory available: 3454MB
- Operating System: Windows XP Service Pack 2.

We set a limit on the time as at most five days (432,000 seconds) for any particular value and stopped the program for any value that took more time that that. We recorded the time and memory required to compute the univariate generating function on this machine in Table 6.1. Observe that the memory utilized grows very fast. We can see for each $n = 4, 5$ there is more than a 10 times increase in the memory utilized when compared to the previous value. On this machine, we were unable to compute the generating function for $n = 6$ due to insufficient memory. This is not surprising, as we would require more than 8000 MB as per the trend.

Table 6.2: Computing the univariate generating function of $2 \times 2 \times n$ solid partitions using the program in Appendix B

n	Time (in Sec.)	Memory (in MB)
1	0	0
2	0	0
3	1	5
4	10	6
5	70	9
6	656	22
7	3301	33
8	20711	68
9	111196	161
10	301674	225

6.2 Optimizations

Due to the high memory utilization by the program in Section 6.1 as shown in Table 6.1, we optimized it in 3 different ways as described below. The limitation in all these optimizations is that, we can compute only the univariate generating function of G_n and $\ell(P_{2\times 2\times n})$, but not $G(n, q, u_{n-1}, v_{n-1}, w_{n-1}, x_{n-1}, u_n, v_n, w_n, x_n)$. All the experiments done below are on the same machine and with the same parameters as described in the end of Section 6.1.

6.2.1 Memoization

On analyzing the recursive procedure calls to our program in Appendix A, we noticed that some procedures were called several times with the same set of arguments. This prompted us to use *memoization*, which is an efficient method to solve problems where procedures with the same set of arguments are called repeatedly. Maple has an internal memoization facility where, using the keyword "options remember" along with the procedure, we can instruct it to store the results of all procedures with their associated parameters in the "remember" table.

To use the storage efficiently, we made some small modifications to our program. Instead of computing the univariate generating function of G_n which is large, we simplified it at each procedure and computed the W-polynomial described in Section 3.2.1, which is smaller. The generating function of G_n can be obtained from the W-polynomial by dividing it with $\prod_{i=1}^{4n}(1-q^i)$. This modified program is shown in Appendix B.

We ran this program and recorded the time and memory required to compute the univariate generating function in Table 6.2. Observe that there is a dramatic improvement in the memory usage when compared to the corresponding values in Table 6.1. The memory grows only about twice

each time. However, the limiting factor now is time. We see that the time required to compute the generating function of G_n grows by about five times each time. The time for $n = 10$ is 301675 seconds, which is about 3.5 days. Based on the trend, we would expect $n = 11$ to take about 15 days which is above the time limit set in Section 6.1 and hence, we did not compute it.

6.2.2 Reduction of variables

In the program in Appendix B, observe that only 4 of the variables are actually modified, the remaining 4 are just passed on to the next level as such. This prompted us to reduce the number of variables in the procedure to compute the generating function of G_n. This also follows from the decomposition steps in the Chapter 5: only the n^{th} layer of G_n is modified and the previous layer is intact.

We modified the program in Appendix B accordingly and the program with reduced variables is shown in Appendix C. This program also uses memoization, hence computes only the W-polynomial. We can obtain the generating function from that as before, by dividing with $\prod_{i=1}^{4n}(1-q^i)$. We ran this program and recorded the time and memory required to compute the univariate generating function of G_n in Table 6.3. We have also shown a few values of the W-polynomial. We can see that, the memory required remains approximately the same, but there is 10-20% improvement in the time taken when compared to the corresponding values in Table 6.2. But the time still grows by almost 5 times and hence for $n = 11$ we would expect it to take about 12 days which is more than our time limit.

6.2.3 Faster computation for $\ell(P_{2\times 2\times n})$

A further optimization can be done if we are only interested in computing the number of linear extensions, $\ell(P_{2\times 2\times n})$. We noted in Section 5.1 in Step 1 from Figure 5.1, that both H_n and H'_n are isomorphic to each other except that H'_n has a strict edge. But the number of linear extensions of the posets represented by these digraphs will be the same, since linear extensions do not vary with strict and non-strict relationships. Therefore, to compute the generating function of H'_n, we just call H_n with the parameters x_n and v_n swapped in all the recursive calls, since these are the two vertices that are reversed in H'_n. Although the resulting generating function is incorrect, we are still guaranteed to get the correct value of $\ell(P_{2\times 2\times n})$ on substituting $q = 1$. Since we use using memoization also, only H_n is computed and H'_n is obtained by just looking up previously stored

Table 6.3: Computing the univariate generating function of $2 \times 2 \times n$ solid partitions using the program in Appendix C

n	$W(P)$	Time (in Sec.)	Mem. (in MB)
1	q^2+1	0	0
2	$q^{16}+2q^{14}+2q^{13}+3q^{12}+3q^{11}+5q^{10}+4q^9+8q^8+4q^7+5q^6+3q^5+3q^4+2q^3+2q^2+1$	0	0
3	$q^{42}+2q^{40}+3q^{39}+5q^{38}+6q^{37}+12q^{36}+14q^{35}+25q^{34}+29q^{33}+41q^{32}+46q^{31}+60q^{30}+68q^{29}+86q^{28}+96q^{27}+117q^{26}+123q^{25}+141q^{24}+137q^{23}+144q^{22}+140q^{21}+144q^{20}+137q^{19}+141q^{18}+123q^{17}+117q^{16}+96q^{15}+86q^{14}+68q^{13}+60q^{12}+46q^{11}+41q^{10}+29q^9+25q^8+14q^7+12q^6+6q^5+5q^4+3q^3+2q^2+1$	2	4
4	$q^{80}+2q^{78}+3q^{77}+6q^{76}+8q^{75}+15q^{74}+21q^{73}+38q^{72}+50q^{71}+78q^{70}+102q^{69}+147q^{68}+189q^{67}+262q^{66}+336q^{65}+451q^{64}+565q^{63}+730q^{62}+888q^{61}+1101q^{60}+1305q^{59}+1571q^{58}+1828q^{57}+2163q^{56}+2476q^{55}+2878q^{54}+3252q^{53}+3695q^{52}+4089q^{51}+4541q^{50}+4906q^{49}+5328q^{48}+5641q^{47}+6008q^{46}+6272q^{45}+6585q^{44}+6781q^{43}+7003q^{42}+7082q^{41}+7164q^{40}+7082q^{39}+7003q^{38}+6781q^{37}+6585q^{36}+6272q^{35}+6008q^{34}+5641q^{33}+5328q^{32}+4906q^{31}+4541q^{30}+4089q^{29}+3695q^{28}+3252q^{27}+2878q^{26}+2476q^{25}+2163q^{24}+1828q^{23}+1571q^{22}+1305q^{21}+1101q^{20}+888q^{19}+730q^{18}+565q^{17}+451q^{16}+336q^{15}+262q^{14}+189q^{13}+147q^{12}+102q^{11}+78q^{10}+50q^9+38q^8+21q^7+15q^6+8q^5+6q^4+3q^3+2q^2+1$	11	6
5	(not shown)	78	10
6	(not shown)	580	22
7	(not shown)	2940	33
8	(not shown)	15848	68
9	(not shown)	84059	148
10	(not shown)	242555	222

Table 6.4: Computation of $\ell(P_{2\times 2\times n})$ using the program in Appendix D

n	$\ell(P)$	Time (in Sec.)	Memory (in MB)
1	2	0	0
2	48	0	0
3	2452	1	4
4	183958	5	5
5	17454844	32	8
6	1941496508	228	14
7	242201554680	1247	25
8	32959299267334	7693	46
9	4801233680739724	38691	105
10	738810565910888784	111789	164
11	118929992674840615128	336722	280

values. So, we can expect the time required to be half of the corresponding values in Table 6.3.

The program with this optimization is shown in Appendix D. We display the time and memory required by this program along with the values of $\ell(P_{2\times 2\times n})$ in Table 6.4. As expected, the time is approximately half the corresponding values in Table 6.3. We are thus able to compute up to $n = 11$ as it is within our time limits. There is also a significant improvement in memory usage when compared to the other three programs.

6.3 Remarks

Even though we have a recurrence for the generating function of $2 \times 2 \times n$ solid partitions, we are able to compute, by automation, the generating function for only a few values of n on a machine with limited resources. But the significance of our work is that, if the recurrence could be solved, we could get a closed form for the generating function. From that, we could also derive a formula to count the linear extensions of the $2 \times 2 \times n$ poset. With the closed form for the generating function of $2 \times 2 \times n$ solid partitions, we could hope to get some insight into the enumeration of $h \times p \times n$ solid partitions.

Chapter 7

Other approaches to enumerate $2 \times 2 \times n$ solid partitions

We derived a recurrence for the generating function of $2 \times 2 \times n$ solid partitions using "digraph rules", that were devised to find integer solutions to systems of linear inequalities of the type $s_a \geq s_b$ and $s_a > s_b$. There are other methods that can be used to find integer solutions to linear inequalities and in this chapter, we study a few of them. We enumerate $2 \times 2 \times n$ solid partitions using these methods and compare their performance, with our program.

7.1 The `Omega` package

In Section 4.1, we mentioned the "five guidelines" as a simplification of MacMahon's *partition analysis* techniques. MacMahon's techniques for partitions analysis viewed partitions as sets of solutions to linear inequalities. He introduced the *Omega* ($\underset{\geq}{\Omega}$) operator, which takes a general form of the generating function of the partitions and eliminates unwanted variables via applications of "Omega rules" to give the required generating function.

Andrews, Paule and Riese [14] investigated MacMahon's partition analysis and noted that using the $\underset{\geq}{\Omega}$ operator to get the generating function of the set of sequences represented by linear inequalities could be automated using computer software. As a result, the `Omega` package, [15] which is a Mathematica package, was built and it was shown in a series of papers [16, 17, 18, 19, 19, 20, 21, 22, 23, 24, 25, 26] that it could be used to compute the generating function of a number of partition

analysis problems. The Omega rules of MacMahon were then used to prove that the solution was correct. The package does not treat families of solid partitions, but rather computes the generating function of a particular solid partition with all the inequalities given. The space complexity of the `Omega` package grows with the number of dimensions of the input and the size of the coefficients of the inequalities. In the case of $2 \times 2 \times n$ solid partitions, the coefficients are all 1, but the number of dimensions increases by four for each each value of n. Hence, we would expect the `Omega` package to perform poorly for higher values of n.

We ran the package in Mathematica 6.0 on the following machine

- Processor: 2.6 GHz AMD Athelon 64x2 dual core processor
- Local memory available: 3454MB
- Operating System: Windows XP Service Pack 2.

and input the inequalities corresponding to $2 \times 2 \times n$ solid partitions. We were not able to compute the generating function for $n > 2$ on this machine due to insufficient memory.

7.2 `LattE`

7.2.1 The package

`LattE` ("Lattice point Enumeration")[27] is a software package used for counting and detecting lattice points inside convex polyhedra and to compute the generating function of these points. It is the first ever implementation of Barvinok's algorithm [28], which is a polynomial time algorithm to count the lattice points in a convex rational polyhedron when the dimensions are fixed. The algorithm uses a method of triangulation to decompose the polyhedron into simplical uni-modular cones, and then find the generating function for each such cone and sum them together.

A polyhedron is defined by a system of linear inequalities given by

$$Ax \leq b \text{ where } A \in \mathbb{Z}^{m \times d}, A = (a_{ij}), \text{ and } b \in \mathbb{Z}^m. \tag{7.1}$$

where $\mathbb{Z}$ is the set of all integers, m is the number of inequalities and d is the number of variables. These inequalities represent the region bounded by a polyhedron P and their solutions represent the set of vertices

Table 7.1: Computation of the generating function of $2 \times 2 \times n$ solid partitions using `LattE`

n	Time (in Sec.)
1	0
2	0
3	1
4	6
5	99
6	$> 86,400$

$$P = \{x \in \mathbb{R}^d : Ax \leq b\}.$$

The goal is to output a generating function

$$f(P; x) = \sum_{\alpha \in P \cap \mathbb{Z}^d} x_1^{\alpha_1} x_2^{\alpha_2} ... x_d^{\alpha_d}.$$

7.2.2 Computing the generating function of $2 \times 2 \times n$ solid partitions using `LattE`

We notice from Equation 7.1 that the system of inequalities that is used to represent a polyhedron is similar to those in Section 2.3.1 used to represent sequences. Thus, we can use `LattE` for our problem, since $2 \times 2 \times n$ solid partitions can be represented as a set of inequalities of the form given in Equation 7.1. This package is written in C for the UNIX operating system and hence could not be run on the same machine described in Section 7.1. We however ran it on a machine with similar configurations given by

- Processor: 2.6 GHz AMD Athelon 64x2 dual core processor
- Local memory available: 3454MB
- Operating System: Red Hat Linux.

For time bounds, we stopped the program if it took significantly more time than the corresponding value of n in Table 6.2.

We display the time taken by `LattE` to compute the generating function of $2 \times 2 \times n$ solid partitions in Table 7.1. The time required by `LattE`, seen in Table 7.1, is comparable to the corresponding values in Table 6.2 for $n \leq 5$, but significantly higher for $n = 6$. Even though `LattE`, promises a polynomial time algorithm for fixed dimensions, it faces its limitation in our case. The number of

dimensions here increases by four with every layer n and so very high dimensions are reached for small values of n.

7.3 The package `RotaStanley`

7.3.1 The method

Zeilberger studied the use of umbral calculus to enumerate P-partitions (defined in Section 2.5) [29]. In this paper, they propose a method of *grafting* where two posets P and Q (labeled naturally from $(1, 2, ...n)$ and $(1, 2, ..., m)$ respectively) that are isomorphic in the last k elements of P and the first k elements of Q, are joined together to form a new poset called the $k-$graft of P and Q. First the labels of Q are promoted by $n-k$ so that the elements are named $(n-k+1, n-k+2, ..., n-k+m)$. The k-graft of P and Q then consists of elements that are the first $n-k$ elements of P and the m elements of Q identified with the new labeling. If the generating functions of the set of P-partitions of P and Q are known, then the generating function of the set of P-partitions of the k-graft of P and Q is found using umbral operators defined in [30].

Application to posets

This method of grafting and using the umbral calculus can be applied to the posets because every poset P can be described (in more than one way) as a chain of elementary posets $(P_1, P_2, ..., P_M)$ together with a sequence of positive integers $(k_1, ..., k_{M-1})$ that describe the interfaces: the last k_1 vertices of P_1 are identified with the first k_1 vertices of P_2 and so on. The resulting poset is denoted by

$$G([P_1, P_2, ..., P_M], [k_1, k_2, ..., k_{M-1}]).$$

To compute the generating function of the set of P-partitions, the poset is viewed as an iterated graft and for each iteration, the methods described above are applied. As long as the interface sizes (k_i's) are not big, the umbral calculus method can be used to compute the generating function of very large posets i.e., as long as the poset is skinny enough, very tall posets can be handled [29].

This method can be used to obtain the generating function for families of posets when the iterative grafting is done with the input poset P itself. The authors have written a Maple package `RotaStanley` [31], which implements their algorithm and have studied several recursive poset fami-

Table 7.2: Computation of generating function of $2 \times 2 \times n$ solid partitions using the package `RotaStanley`

n	Time (in Sec.)	Memory (in MB)
1	-	-
2	10	8
3	14	11
4	21	21
5	33	164
6	-	$> 1,800$

lies.

7.3.2 Computing the generating function of $2 \times 2 \times n$ solid partitions using the package `RotaStanley`

The authors have studied the $2 \times 2 \times n$ poset and have a written a separate function to compute the generating function of its set of P-partitions. We mentioned in Section 3.1.3 that the generating function of $h \times p \times n$ solid partitions is same as the generating function of the set of P-partitions of the corresponding poset. We ran `RotaStanley` in Maple 11, on the machine described in Section 7.1 and display the time and memory required to compute the generating function of $2 \times 2 \times n$ solid partition in Table 7.2. Although the time required to compute the generating function using `RotaStanley` in Table 7.2 is comparable to values in Table 6.2, we can see that the memory requirements are higher. We were not able to compute the generating function for $n = 6$ as the memory required was greater than the available memory on the system. This is surprising, as the $2 \times 2 \times n$ poset is "skinny" (has a width of 4), yet our machine is not able to compute higher values of the generating function using the package.

7.4 The Combinatorial Object Server (COS)

7.4.1 The package

The Combinatorial Object Server is a mathematical tool [32] that can generate different combinatorial objects like permutations, combinations, different tree types, unlabeled graphs, linear extensions, ideals and many more. Of particular interest to us, is the program that generates and counts linear extensions. Based on the type of listing requested, two different algorithms are implemented. If Gray code output is requested, the program implements Ruskeys's algorithm to generate and count

Table 7.3: Using the Combinatorial Object Server to count the linear extensions of the $2 \times 2 \times n$ poset

n	Time (in Sec.)
1	0
2	0
3	0
4	0
5	0
6	11
7	1278
8	$> 86,400$

linear extensions.

Pruesse and Ruskey developed a method of generating linear extensions based on the idea of listing the successive elements through transpositions [33, 34, 35]. They developed an algorithm that takes constant amortized time, to generate each linear extension ($O(\ell(P))$). To count the linear extensions, the program is modified accordingly to save some computations and count them efficiently. Their algorithm extends the practical range of posets for which this can be done, more than other previous algorithms.

7.4.2 Counting linear extensions of the $2 \times 2 \times n$ poset using the COS

We input the $2 \times 2 \times n$ poset to the Combinatorial Object Server to count its linear extensions, and recorded the time taken for each values of n in Table 7.3. It is a program in C, so we ran it on the UNIX machine described in Section 7.2.2. When compared to our program to count linear extensions in Table 6.4, we can see that it is faster up to $n = 6$, comparable for $n = 7$ and slows down for $n \geq 8$. It is beyond the time bounds set in Section 7.2.2 and hence we did not compute it. Even though the program is optimized for counting, it still runs in $O(\ell(P))$ and seeing the growth of $\ell(P_{2\times 2\times n})$ from Table 6.4, this slowdown is not surprising.

7.5 The `Posets` package

7.5.1 The package

Stembridge developed a software package in Maple that consists of about 40 programs that provide an environment for computations involving partially ordered sets [36]. Among the various programs, the one that is of interest to us is the one that computes the W-polynomial (defined in Section 3.2.1)

and the number of linear extensions of a given poset. These algorithms run in time proportional to the number of order ideals (defined in Section 2.4.4), $N_{J(P)}$, of a poset P. For any poset P, $N_{J(P)}$ is in $O(2^n)$ which is asymptotically lower than the number of linear extensions, $\ell(P)$, which is in $O(n!)$. So we can expect these algorithms to be much faster than any of the other methods described before. The number of order ideals $N_{J(P_{2\times 2\times n})}$ of the $2 \times 2 \times n$ poset is given by

$$N_{J(P_{2\times 2\times n})} = \frac{n^2(n^2-1)}{12}.$$

Computing the W-polynomial

Stembridge implements two algorithms to compute the W-polynomial based on the size of the poset. The first computes the W-polynomial if the size of the poset P is less than 6. It has minimal space requirements and runs in time, linear in the number of linear extensions of P. It is a straightforward algorithm that uses Equation 3.6 to compute the descent set for each of the linear extensions.

If the size of the poset is greater than 6, a different algorithm whose running time is quadratic in $N_{J(P)}$ is used. This algorithms involves formation of the order ideal poset $J(P)$ and then computing some functions on the maximal chains (defined in Section 2.4.5) in $J(P)$, to compute the W-polynomial [37].

Counting Linear Extensions

To count the number of linear extensions, a faster algorithm called the "voodoo" method is used. This is an application of the result that the number of linear extensions of a poset P is equal to the number of chains in the order ideal $J(P)$ [5]. It is optimized further by using dynamic programming. Once the order ideal $J(P)$ is computed, the algorithm computes the number of chains in $J(P)$. If we think of $J(P)$ as a graph, the number of chains is the same as the number of paths from the top most element to the bottom most element. The complexity is linear in the number of order ideals of P.

7.5.2 Using the `Posets` package on the $2 \times 2 \times n$ poset

We ran the `Posets` package in Maple 11, on the machine described in Section 7.1 and computed the W-polynomial for the $2 \times 2 \times n$ poset. The generating function of $2 \times 2 \times n$ solid partitions can be obtained from the W-polynomial by dividing it by $\prod_{i=1}^{4n}(1-q^i)$ (refer Equation 3.4). The time and

Table 7.4: Computation of W-polynomial of the $2 \times 2 \times n$ poset using `posets`

n	Time (in Sec.)	Memory (in MB)
1	0	0
2	0	0
3	0	0
4	0	0
5	0	3
6	0	4
7	1	4
8	1	5
9	1	6
10	3	9
11	6	16
12	18	25
13	44	41
14	95	63
15	179	103
16	313	172
17	515	243
18	781	380
19	1194	553
20	1707	726

memory required to compute the generating function are displayed in Table 7.4. As expected, the time required to compute the W-polynomial is much lower than any of the methods described before including all our programs. We are able to compute much higher values of W-polynomial using this package in a shorter time. The time taken in Table 7.4 grows by less than three times for each value of n. Counting linear extensions is even faster. Table 7.5 shows the number of linear extensions of the $2 \times 2 \times n$ poset computed for values of n between 12 and 30 using the `Posets` package. The time taken to compute these was less than 20 seconds and the memory utilized was less than 20MB.

7.6 Remarks

On using these packages to enumerate $2 \times 2 \times n$ solid partitions, we are able to see their advantages and limitations. The main advantage of these packages over our program is that, they are not designed for just one special case. That is also a disadvantage, since they do not have any special knowledge about our particular problem, unlike our program. The `Omega` package and `LattE`, like the "digraph methods", treat solid partitions as integer solutions to sets of linear inequalities. The `Omega` package is a powerful tool that was used to solve a number of partition analysis problems. Since its space complexity grows with the number of dimensions of the input, it is not very useful for

Table 7.5: Values of number of linear extensions of the $2 \times 2 \times n$ poset computed using `posets`

n	$\ell(G_n)$
12	19880920716640427983476
13	34316244822273802730567 28
14	60888041987358651566956 4728
15	11065401619133834134667 0548240
16	20536574090713344110860 752530646
17	38829250243311747968571 01684510428
18	74641093144894501219651 3727291312844
19	14562636267080576026480 941424305761 6552
20	28794547473359904233269297596817899967540
21	57629311822629267879489462597213460993374 48
22	11661853959475744202290003661638574683371 48200
23	23838098130251086240658605190128959763200 3118992
24	49180740964278427332002617903561118285268230978484
25	10233409402065855596885427205182450024540045694903496
26	21461794862098971035406912983541782420486914174822620 72
27	45340014425653478013912136776165754814547787522304292 7920
28	96436903908611647069262069806690816719553841382241397251848
29	20641908397438606421142576358642139410375633787282792348176176
30	44444733564270124952619653420581388669360072357092580195847515 68

$2 \times 2 \times n$ solid partitions, since the number of dimensions here, increases by four for every value of n. `LattE` has the same disadvantage with respect to our problem. The advantage of `LattE` over the `Omega` package is that, the time complexity of `LattE` does not increase with size of the coefficients for a fixed dimension, unlike the `Omega` package. Hence `LattE` can be used to solve linear inequalities with high value of coefficients faster than the `Omega` package.

All the other packages that we used, focus on the poset aspect of the problem. The umbral calculus method of Zeilberger, views $2 \times 2 \times n$ solid partitions as the set of P-partitions of the $2 \times 2 \times n$ poset and computes its generating function. The umbral calculus method could be used on large and "skinny" posets, and also on poset families, to derive an "umbral recurrence". However, our machine running the Maple implementation, `RotaStanley`, could not compute the generating function for $n = 6$ due to high memory requirements. The Combinatorial Object Server for counting linear extensions is fast for small values of n, but slows down for higher values because of the $O(\ell(P))$ algorithm.

The `Posets` package also does computations on posets and can compute the W-polynomial of the poset corresponding to $2 \times 2 \times n$ solid partitions. This is the only package that, among the others we used, outperforms our program. It exploits some of the basic properties of the poset, and combined with some clever methods, gives a $O(N_{J(P)})$ algorithm to compute the W-polynomial and count the

linear extensions. The limitation of this method is that we can never get the W-polynomial for the general case n, unlike our recurrence. Each poset has to be explicitly input, which could be tedious for very high values of n.

Chapter 8

Further observations and future work

In this chapter, we present some interesting observations we made about $2 \times 2 \times n$ solid partitions in this thesis and some possible scope for future work.

8.1 Further observations

There are a few interesting observations about $2 \times 2 \times n$ solid partitions and the W-polynomial of the $2 \times 2 \times n$ poset that we made using the "digraph methods", `Posets` package and the Sloan's encyclopedia. Let $d(n)$ be the degree of the W-polynomial of the $2 \times 2 \times n$ poset, then

1. The values of $d(n)$, follow the sequence $2, 16, 42, 80, 130, 192, 266, ...$ which can be generated as (Sequence No. A139267 in [1])

$$d(n) = 2n(3n - 2).$$

2. The sequence formed by the coefficients of the W-polynomial is symmetric i.e.,

$$\text{coeff}(q^k) = \text{coeff}(q^{(d(n)-k)}).$$

3. In the W-polynomial of $2 \times 2 \times n$ poset,

$$\text{For } 0 \leq i \leq d(n),\ \{\text{coeff}(q^i) = 0\} \Leftrightarrow \{i = 0 \text{ or } i = d(n) - 1\}.$$

4. The number of linear extensions of the poset represented by the digraph H_n and H'_n from Figure 5.1 are the same and equal to $\frac{\ell(P_{2\times 2\times n})}{2}$.

5. The following identity holds for the reduced variable version of our recurrence:

$$H'(n, q, q, q, q, q) = H(n, 1/q, 1/q, 1/q, 1/q, 1/q) * \frac{1}{(q^{d(n)+2})}.$$

   It provides another way to compute the generating function of $H'(n)$ given the generating function of $H(n)$.

6. The number of order ideals $N_{J(P_{2\times 2\times n})}$ of the $2 \times 2 \times n$ poset follows the sequence 6,20,50,105,... called the "4-dimensional pyramidal numbers" and are generated by (Sequence No. A002415 in [1])

$$N_{J(P_{2\times 2\times n})} = \frac{n^2(n^2 - 1)}{12}$$

   where $n \geq 3$

7. The number of sub-posets of size $m/2$ in the order ideal poset $J(P_{2\times 2\times n})$ of the $2 \times 2 \times n$ poset , (where $m = 4n$) follows the sequence 2,4,8,13,20,30,... (Sequence No. A061866 in [1]) which is the number of solutions to

$$x + y + z = 0 \ mod\ 3, \text{ where } 1 \leq x \leq y \leq z \leq m.$$

8. The number of sub-posets of sizes $\frac{m}{2} - 1$ and $\frac{m}{2} + 1$ in $J(P_{2\times 2\times n})$, are the same and follow the sequence 1,3,6,11,18,27,39,... (Sequence No. A014125 in [1]).

9. If we split the digraph representing $2 \times 2 \times n$ solid partitions along the plane shown in Figure 8.1 we get two graphs that represent $3 \times n$ plane partitions and the generating function for that can be derived from Equation 3.1 as

$$\frac{1}{(q;q)_n(q;q^2)_n(q;q^3)_n}.$$

However, if we split as shown in Figure 8.2, we get 2 graphs that look like a $3 \times n$ spine shaped graph. There is no known closed form of the generating function of that digraph. In 1965 Krewaras proved that the formula

$$\frac{4^n(3n)!}{(n+1)!(2n+1)!}$$

counts the number of planar lattice walks of length $3n$ starting and ending at (0,0), remaining in the first quadrant, and using only NE,W,S steps. This proof is considered to be difficult [38]. It can be shown that the number of linear extensions of the poset represented by the graph shown in Figure 8.2 can be given by the same formula.

8.2 Conclusion

In this thesis, we are successful in computing an explicit recurrence for the generating function of $2 \times 2 \times n$ solid partitions. We implemented the recurrence in Maple, optimized using memoization and reduction of variables and computed the generating function for a few values of n. We also used other packages like `LattE`, `RotaStanley`, `Omega` package, COS and `Posets` package to enumerate $2 \times 2 \times n$ solid partitions and count the linear extensions of the $2 \times 2 \times n$ poset. We observed that our program performs better than all these packages except the `Posets` package, which is much faster.

8.3 Future Work

This work presents some scope for future work. We list some possible directions for future work below.

1. Solve the recurrence for $2 \times 2 \times n$ solid partitions and get the closed form of the counting generating function.
2. Use the digraph methods to derive the counting generating function of other solid partitions like $3 \times 2 \times n$, and eventually extend it to get a generalized form of the generating function for $h \times p \times n$ solid partitions.
3. In Section 8.1, we mentioned the formula to count the number of linear extensions of the $3 \times n$ spine graph shown in Figure 8.2. Since the number of linear extensions has a nice form, the generating function of the set of graph too could have a closed form.

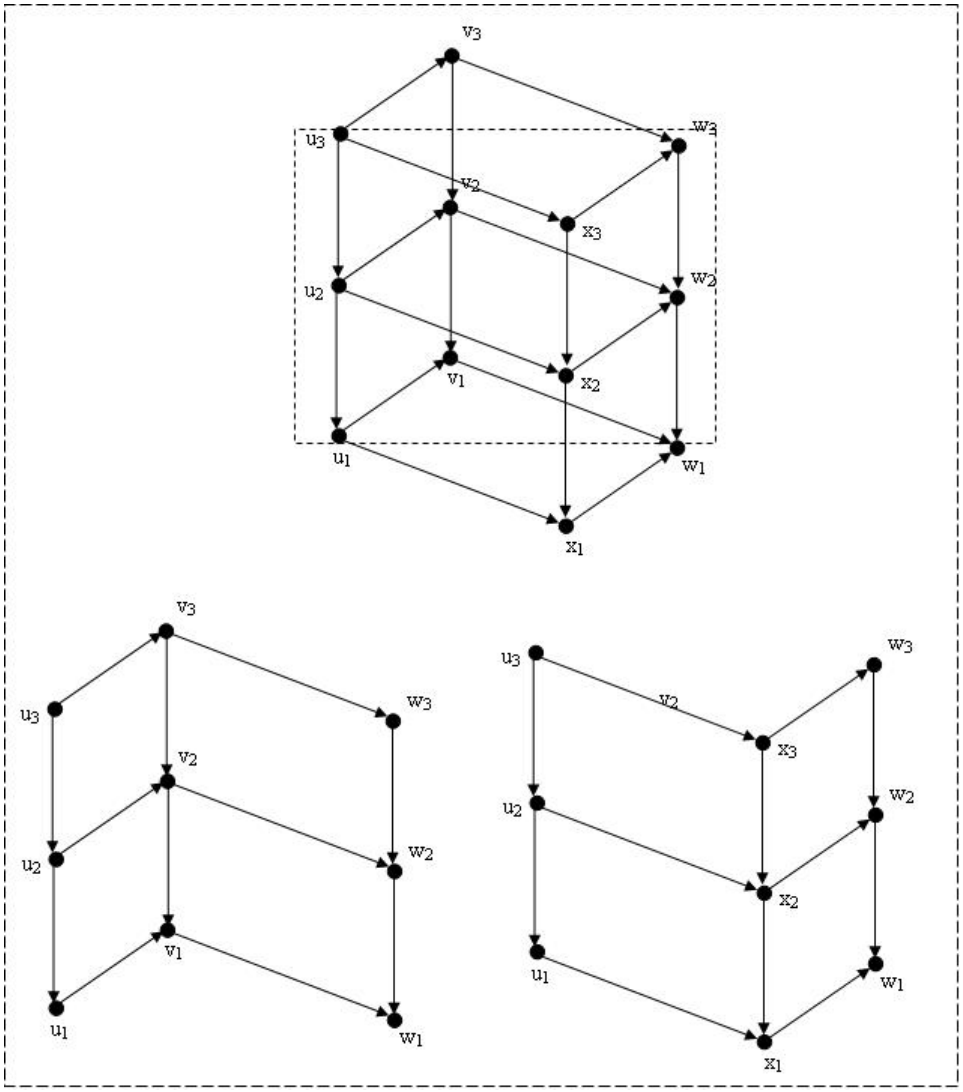

Figure 8.1: Splitting G_n one way

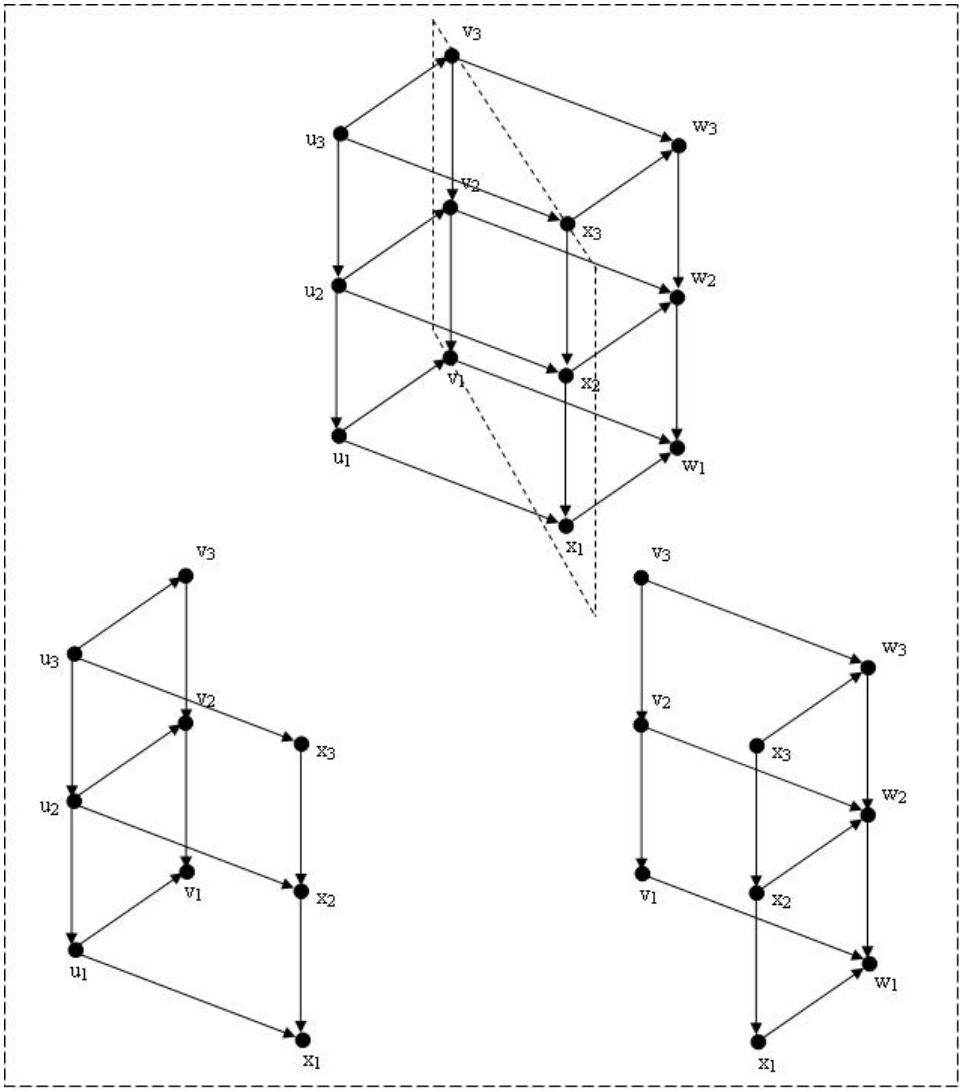

Figure 8.2: Another splitting of G_n

Bibliography

[1] N.J.A Sloane. Online Encyclopedia of Integer Sequences. World Wide Web, 2007.

[2] William J Davis, Erwin D'Souza, Sunyoung Lee, and Carla D. Savage. Enumeration of integer solutions to linear inequalities defined by digraphs. In *Contemporary Mathematics*, volume 452, pages 79–91. American Mathematical Society, 2008.

[3] Sylvie Corteel, Sunyoung Lee, and Carla D. Savage. Five guidelines for partition analysis with applications to lecture hall-type theorems. In *Combinatorial number theory*, pages 131–155. de Gruyter, Berlin, 2007.

[4] Richard P. Stanley. *Enumerative combinatorics. Vol. 1*, volume 49 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 1997. With a foreword by Gian-Carlo Rota, Corrected reprint of the 1986 original.

[5] Richard P. Stanley. *Ordered structures and partitions*. American Mathematical Society, Providence, R.I., 1972. Memoirs of the American Mathematical Society, No. 119.

[6] Percy A. MacMahon. *Combinatory analysis*. Two volumes (bound as one). Chelsea Publishing Co., New York, 1960.

[7] Lorne Houten. A note on solid partitions. *Acta Arith.*, 15:71–76, 1968.

[8] Ville Mustonen and R. Rajesh. Numerical estimation of the asymptotic behaviour of solid partitions of an integer. *J. Phys. A*, 36(24):6651–6659, 2003.

[9] Graham Brightwell and Peter Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.

[10] T. Kyle Petersen. Enriched P-partitions and peak algebras. *Adv. Math.*, 209(2):561–610, 2007.

[11] John R. Stembridge. Enriched P-partitions. *Trans. Amer. Math. Soc.*, 349(2):763–788, 1997.

[12] Donald E. Knuth. A note on solid partitions. *Math. Comp.*, 24:955–961, 1970.

[13] Erwin D'Souza. Automating the enumeration of integer sequences defined by directed graphs. Master's thesis, North Carolina State University, 2005.

[14] George E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis: the Omega package. *European J. Combin.*, 22(7):887–904, 2001.

[15] George E. Andrews, Peter Paule, and Axel Riese. `Omega` package. `http://www.risc.uni-linz.ac.at/research/combinat/software/Omega/`.

[16] George E. Andrews. MacMahon's partition analysis. I. The lecture hall partition theorem. In *Mathematical essays in honor of Gian-Carlo Rota (Cambridge, MA, 1996)*, volume 161 of *Progr. Math.*, pages 1–22. Birkhäuser Boston, Boston, MA, 1998.

[17] George E. Andrews. MacMahon's partition analysis. II. Fundamental theorems. *Ann. Comb.*, 4(3-4):327–338, 2000. Conference on Combinatorics and Physics (Los Alamos, NM, 1998).

[18] George E. Andrews and Peter Paule. MacMahon's partition analysis. IV. Hypergeometric multisums. *Sém. Lothar. Combin.*, 42:Art. B42i, 24 pp. (electronic), 1999. The Andrews Festschrift (Maratea, 1998).

[19] George E. Andrews, Peter Paule, Axel Riese, and Volker Strehl. MacMahon's partition analysis. V. Bijections, recursions, and magic squares. In *Algebraic combinatorics and applications (Gößweinstein, 1999)*, pages 1–39. Springer, Berlin, 2001.

[20] George E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis. VI. A new reduction algorithm. *Ann. Comb.*, 5(3-4):251–270, 2001. Dedicated to the memory of Gian-Carlo Rota (Tianjin, 1999).

[21] George E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis. VII. Constrained compositions. In *q-series with applications to combinatorics, number theory, and physics (Urbana, IL, 2000)*, volume 291 of *Contemp. Math.*, pages 11–27. Amer. Math. Soc., Providence, RI, 2001.

[22] George E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis. VIII. Plane partition diamonds. *Adv. in Appl. Math.*, 27(2-3):231–242, 2001. Special issue in honor of Dominique Foata's 65th birthday (Philadelphia, PA, 2000).

[23] George E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis. IX. k-gon partitions. *Bull. Austral. Math. Soc.*, 64(2):321–329, 2001.

[24] G. E. Andrews, Peter Paule, and Axel Riese. MacMahon's partition analysis. X. Plane partitions with diagonals. *South East Asian J. Math. Math. Sci.*, 3(1):3–14, 2004.

[25] George E. Andrews and Peter Paule. MacMahon's partition analysis. XI. Broken diamonds and modular forms. *Acta Arith.*, 126(3):281–294, 2007.

[26] George E. Andrews and Peter Paule. MacMahon's partition analysis. XII. Plane partitions. *J. Lond. Math. Soc. (2)*, 76(3):647–666, 2007.

[27] Jesús A. De Loera, David Haws, Raymond Hemmecke, Peter Huggins, Jeremiah Tauzer, and Ruriko Yoshida. `LattE`. `http://www.math.ucdavis.edu/~latte/`.

[28] Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4):769–779, 1994.

[29] Shalosh B. Ekhad and Doron Zeilberger. Using rota's umbral calculus to enumerate stanley's p-partitions. In *Advances in Applied Mathematics*. Elsevier Inc., 2007.

[30] Steven M. Roman and Gian-Carlo Rota. The Umbral calculus. *Advances in Math.*, 27(2):95–188, 1978.

[31] Shalosh B. Ekhad and Doron Zeilberger. `RotaStanley`. `http://www.math.rutgers.edu/~zeilberg/tokhniot/RotaStanley`.

[32] Frank Ruskey. Combinatorial object server. http://theory.cs.uvic.ca/root.html.

[33] Gara Pruesse and Frank Ruskey. Generating the linear extensions of certain posets by transpositions. *SIAM J. Discrete Math.*, 4(3):413–422, 1991.

[34] Frank Ruskey. Generating linear extensions of posets by transpositions. *J. Combin. Theory Ser. B*, 54(1):77–101, 1992.

[35] Gara Pruesse and Frank Ruskey. Generating linear extensions fast. *SIAM J. Comput.*, 23(2):373–386, 1994.

[36] John R. Stembridge. `Posets` package. `http://www.math.lsa.umich.edu/~jrs/maple.html#posets`.

[37] John R. Stembridge. Counterexamples to the poset conjectures of Neggers, Stanley, and Stembridge. *Trans. Amer. Math. Soc.*, 359(3):1115–1128 (electronic), 2007.

[38] Mireille Bousquet-Melou. Walks in the quarter plane: Kreweras' algebraic model. *The Annals of Applied Probability*, 15:1451, 2005.

Appendices

Appendix A

Program without optimization

```
Gn := proc(n,q,a,b,c,d,u,v,w,x);
      return (Hn(n,q,a,b,c,d,u,v,w,x)+Hn1(n,q,a,b,c,d,u,v,w,x))

end:
Hn := proc(n,q,a,b,c,d,u,v,w,x);
      if n=1 then
       return (((1)/((1-u)*(1-u*x)*(1-u*v*x)*(1-u*v*w*x))))
      else
       return (Kn(n,q,a,b,c,d,u,v,w,x)-
       Pn(n,q,a,b,c,d,u,v,w,x))
      fi;
end:
Kn := proc(n,q,a,b,c,d,u,v,w,x) ;
      return ( -(Gn(n-1,q,q,q,q,q,a*u,v*b*x*w,c,d)*w)/
        ((1-u)*(1-v*x*w)*(1-x)*(1-w))
      +((Gn(n-1,q,q,q,q,q,a*u,v*b*x,c*w,d)))/
        ((1-u)*(1-v*x)*(1-x)*(1-w))
      +((Gn(n-1,q,q,q,q,q,a*u*x,v*b*w,c,d))*x*w)/
        ((1-u*x)*(1-v*w)*(1-w)*(1-x))
```

```
      -((Gn(n-1,q,q,q,q,q,a*u*x,v*b,c*w,d))*x)/
        ((1-u*x)*(1-v)*(1-x)*(1-w))
      -((Gn(n-1,q,q,q,q,q,a*u*x*v*w,b,c,d))*x*(v^2)*(w^3))/
        ((1-u*x*v*w)*(1-x*v*w)*(1-v*w)*(1-w))
      +((Gn(n-1,q,q,q,q,q,a*u*x*v,b,c*w,d))*x*(v^2))/
        ((1-u*v*x)*(1-x*v)*(1-v)*(1-w)))
end:
Pn := proc(n,q,a,b,c,d,u,v,w,x);
      return ( (Dn(n,q,a*u,b,c*w,d,u,v,w,x)-
      (Dn(n,q,a*u,b,c,d,u,v*w,w,x)*w))/((1-u)*(1-w)) )
end:
Dn := proc(n,q,a,b,c,d,u,v,w,x);
      return ( (Hn(n-1,q,q,q,q,q,a,b,c,v*x*d)*v)/
        ((1-v)*(1-x*v))
      +(Hn(n-1,q,q,q,q,q,a,b*v*x,c,d))/
        ((1-v*x)*(1-x))
      -(Hn(n-1,q,q,q,q,q,a,b*v,c,d*x))/
        ((1-v)*(1-x)))
end:

Hn1 := proc(n,q,a,b,c,d,u,v,w,x);
       if n=1 then
        return ( (v*u)/((1-u)*(1-u*v)*(1-u*v*x)*(1-u*v*w*x)) )
       else
        return ( Kn1(n,q,a,b,c,d,u,v,w,x)-
        Pn1(n,q,a,b,c,d,u,v,w,x) )
       fi;
end:
Kn1 := proc(n,q,a,b,c,d,u,v,w,x);
       return (-((Gn(n-1,q,q,q,q,q,a*u,b,c,d*x*v*w))*v*w)/
         ((1-u)*(1-x*v*w)*(1-v)*(1-w))
       +((Gn(n-1,q,q,q,q,q,a*u*v,b,c,d*x*w))*w*v)/
```

```
          ((1-u*v)*(1-x*w)*(1-v)*(1-w))
        -((Gn(n-1,q,q,q,q,q,a*u*v*x*w,b,c,d))*v*x*(w^2))/
          ((1-x*u*v*w)*(1-x*w*v)*(1-x*w)*(1-w))
        +((Gn(n-1,q,q,q,q,q,a*u,b,c*w,d*x*v))*v)/
          ((1-u)*(1-x*v)*(1-v)*(1-w))
        -((Gn(n-1,q,q,q,q,q,a*u*v,b,c*w,d*x))*v)/
          ((1-u*v)*(1-x)*(1-v)*(1-w))
        +((Gn(n-1,q,q,q,q,q,a*u*v*x,b,c*w,d))*v*x)/
          ((1-u*v*x)*(1-v*x)*(1-x)*(1-w)) )
end:
Pn1 := proc(n,q,a,b,c,d,u,v,w,x);
        return ((Dn1(n,q,a*u,b,c*w,d,u,v,w,x)-
        (Dn1(n,q,a*u,b,c,d,u,v,w,x*w)*w))/((1-u)*(1-w)))
end:
Dn1 := proc(n,q,a,b,c,d,u,v,w,x) ;
        return (((Hn1(n-1,q,q,q,q,q,a,b*v*x,c,d)))/
          ((1-v*x)*(1-x))
        +((Hn1(n-1,q,q,q,q,q,a,b,c,d*x*v))*v)/
          ((1-x*v)*(1-v))
        -((Hn1(n-1,q,q,q,q,q,a,b*v,c,d*x)))/
          ((1-x)*(1-v)) )
end:
```

Appendix B

Program with memoization

```
Gn := proc(n,q,a,b,c,d,u,v,w,x) options remember;
      return (Hn(n,q,a,b,c,d,u,v,w,x)+Hn1(n,q,a,b,c,d,u,v,w,x))
end:

Hn := proc(n,q,a,b,c,d,u,v,w,x) options remember;
      if n=1 then
       return simplify(((1)/
        ((1-u)*(1-u*x)*(1-u*v*x)*(1-u*v*w*x)))
          *product('1-q^i','i'=1..4))
      else
       return simplify((
         (-(Gn(n-1,q,q,q,q,q,a*u,v*b*x*w,c,d)*w)/
           ((1-u)*(1-v*x*w)*(1-x)*(1-w))
         +((Gn(n-1,q,q,q,q,q,a*u,v*b*x,c*w,d)))/
           ((1-u)*(1-v*x)*(1-x)*(1-w))
         +((Gn(n-1,q,q,q,q,q,a*u*x,v*b*w,c,d))*x*w)/
           ((1-u*x)*(1-v*w)*(1-w)*(1-x))
         -((Gn(n-1,q,q,q,q,q,a*u*x,v*b,c*w,d))*x)/
           ((1-u*x)*(1-v)*(1-x)*(1-w))
```

```
          -((Gn(n-1,q,q,q,q,q,a*u*x*v*w,b,c,d))*x*(v^2)*(w^3))/
            ((1-u*x*v*w)*(1-x*v*w)*(1-v*w)*(1-w))
          +((Gn(n-1,q,q,q,q,q,a*u*x*v,b,c*w,d))*x*(v^2))/
            ((1-u*v*x)*(1-x*v)*(1-v)*(1-w)))

          -((((Hn(n-1,q,q,q,q,q,a*u,b,c*w,v*x*d)*v)/((1-v)*(1-x*v))
          +(Hn(n-1,q,q,q,q,q,a*u,b*v*x,c*w,d))/((1-v*x)*(1-x))
          -(Hn(n-1,q,q,q,q,q,a*u,b*v,c*w,d*x))/((1-v)*(1-x)))

          -(((Hn(n-1,q,q,q,q,q,a*u,b,c,v*w*x*d)*v*w)/((1-v*w)*(1-x*v*w))
          +(Hn(n-1,q,q,q,q,q,a*u,b*v*w*x,c,d))/((1-v*w*x)*(1-x))
          -(Hn(n-1,q,q,q,q,q,a*u,b*v*w,c,d*x))/
            ((1-v*w)*(1-x)))*w))/((1-u)*(1-w)) ))
              *product('1-q^i','i'=(4*(n-1)+1)..4*n))
      fi;
end:

Hn1 := proc(n,q,a,b,c,d,u,v,w,x) options remember;
       if n=1 then
        return simplify(( (v*u)/
          ((1-u)*(1-u*v)*(1-u*v*x)*(1-u*v*w*x)) )
            *product('1-q^i','i'=1..4))
       else
        return simplify
          (((-((Gn(n-1,q,q,q,q,q,a*u,b,c,d*x*v*w))*v*w)/
            ((1-u)*(1-x*v*w)*(1-v)*(1-w))
          +((Gn(n-1,q,q,q,q,q,a*u*v,b,c,d*x*w))*w*v)/
            ((1-u*v)*(1-x*w)*(1-v)*(1-w))
          -((Gn(n-1,q,q,q,q,q,a*u*v*x*w,b,c,d))*v*x*(w^2))/
            ((1-x*u*v*w)*(1-x*w*v)*(1-x*w)*(1-w))
          +((Gn(n-1,q,q,q,q,q,a*u,b,c*w,d*x*v))*v)/
            ((1-u)*(1-x*v)*(1-v)*(1-w))
```

```
        -((Gn(n-1,q,q,q,q,q,a*u*v,b,c*w,d*x))*v)/
          ((1-u*v)*(1-x)*(1-v)*(1-w))
        +((Gn(n-1,q,q,q,q,q,a*u*v*x,b,c*w,d))*v*x)/
          ((1-u*v*x)*(1-v*x)*(1-x)*(1-w)) )

        -(((((Hn1(n-1,q,q,q,q,q,a*u,b*v*x,c*w,d)))/((1-v*x)*(1-x))
        +((Hn1(n-1,q,q,q,q,q,a*u,b,c*w,d*x*v))*v)/((1-x*v)*(1-v))
        -((Hn1(n-1,q,q,q,q,q,a*u,b*v,c*w,d*x)))/((1-x)*(1-v)) )

        -((((Hn1(n-1,q,q,q,q,q,a*u,b*v*x*w,c,d)))/((1-v*x*w)*(1-x*w))
        +((Hn1(n-1,q,q,q,q,q,a*u,b,c,d*x*w*v))*v)/((1-x*w*v)*(1-v))
        -((Hn1(n-1,q,q,q,q,q,a*u,b*v,c,d*x*w)))/((1-x*w)*(1-v)) )*w))/
          ((1-u)*(1-w))) )*product('1-q^i','i'=(4*(n-1)+1)..4*n))
      fi;
end:
```

Appendix C

Program with memoization and reduced variable optimization

```
Gn := proc(n,q,u,v,w,x) options remember;
      return Hn(n,q,u,v,w,x)+Hn1(n,q,u,v,w,x);

end:
Hn := proc(n,q,u,v,w,x) options remember;
      if n=1 then
       return simplify(((1)/((1-u)*(1-u*x)*(1-u*v*x)*(1-u*v*w*x)))
         *product('1-q^i','i'=1..4))
      else
       return simplify((
       (-simplify((Gn(n-1,q,q*u,v*q*x*w,q,q)*w)/
         ((1-u)*(1-v*x*w)*(1-x)*(1-w))
           *product('1-q^i','i'=(4*(n-1)+1)..4*n))
       +simplify(((Gn(n-1,q,q*u,v*q*x,q*w,q)))/
         ((1-u)*(1-v*x)*(1-x)*(1-w))
           *product('1-q^i','i'=(4*(n-1)+1)..4*n))
```

```
        +simplify(((Gn(n-1,q,q*u*x,v*q*w,q,q))*x*w)/
          ((1-u*x)*(1-v*w)*(1-w)*(1-x))
            *product('1-q^i','i'=(4*(n-1)+1)..4*n))
        -simplify(((Gn(n-1,q,q*u*x,v*q,q*w,q))*x)/
          ((1-u*x)*(1-v)*(1-x)*(1-w))
            *product('1-q^i','i'=(4*(n-1)+1)..4*n))
        -simplify(((Gn(n-1,q,q*u*x*v*w,q,q,q))*x*(v^2)*(w^3))/
          ((1-u*x*v*w)*(1-x*v*w)*(1-v*w)*(1-w))
            *product('1-q^i','i'=(4*(n-1)+1)..4*n))
        +simplify(((Gn(n-1,q,q*u*x*v,q,q*w,q))*x*(v^2))/
          ((1-u*v*x)*(1-x*v)*(1-v)*(1-w))
            *product('1-q^i','i'=(4*(n-1)+1)..4*n)))

        -(((simplify((Hn(n-1,q,q*u,q,q*w,v*x*q)*v)/
          ((1-v)*(1-x*v))*product('1-q^i','i'=(4*(n-1)+1)..4*n))
        +simplify((Hn(n-1,q,q*u,q*v*x,q*w,q))/
          ((1-v*x)*(1-x))*product('1-q^i','i'=(4*(n-1)+1)..4*n))
        -simplify((Hn(n-1,q,q*u,q*v,q*w,q*x))/
          ((1-v)*(1-x))*product('1-q^i','i'=(4*(n-1)+1)..4*n)))
        -((simplify((Hn(n-1,q,q*u,q,q,v*w*x*q)*v*w)/
          ((1-v*w)*(1-x*v*w))*product('1-q^i','i'=(4*(n-1)+1)..4*n))
        +simplify((Hn(n-1,q,q*u,q*v*w*x,q,q))/
          ((1-v*w*x)*(1-x))*product('1-q^i','i'=(4*(n-1)+1)..4*n))
        -simplify((Hn(n-1,q,q*u,q*v*w,q,q*x))/
          ((1-v*w)*(1-x))
            *product('1-q^i','i'=(4*(n-1)+1)..4*n)))*w))/
              ((1-u)*(1-w)))))
      fi;
end:

Hn1 := proc(n,q,u,v,w,x) options remember;
       if n=1 then
```

```
 return simplify(( (v*u)/
   ((1-u)*(1-u*v)*(1-u*v*x)*(1-u*v*w*x)) )
     *product('1-q^i','i'=1..4))
else
 return simplify(
   ((-((Gn(n-1,q,q*u,q,q,q*x*v*w))*v*w)/
     ((1-u)*(1-x*v*w)*(1-v)*(1-w))
   +((Gn(n-1,q,q*u*v,q,q,q*x*w))*w*v)/
     ((1-u*v)*(1-x*w)*(1-v)*(1-w))
   -((Gn(n-1,q,q*u*v*x*w,q,q,q))*v*x*(w^2))/
     ((1-x*u*v*w)*(1-x*w*v)*(1-x*w)*(1-w))
   +((Gn(n-1,q,q*u,q,q*w,q*x*v))*v)/
     ((1-u)*(1-x*v)*(1-v)*(1-w))
   -((Gn(n-1,q,q*u*v,q,q*w,q*x))*v)/
     ((1-u*v)*(1-x)*(1-v)*(1-w))
   +((Gn(n-1,q,q*u*v*x,q,q*w,q))*v*x)/
     ((1-u*v*x)*(1-v*x)*(1-x)*(1-w)) )

   -(((((Hn1(n-1,q,q*u,q*v*x,q*w,q)))/((1-v*x)*(1-x))
   +((Hn1(n-1,q,q*u,q,q*w,q*x*v))*v)/((1-x*v)*(1-v))
   -((Hn1(n-1,q,q*u,q*v,q*w,q*x)))/((1-x)*(1-v)) )
   -((((Hn1(n-1,q,q*u,q*v*x*w,q,q)))/((1-v*x*w)*(1-x*w))
   +((Hn1(n-1,q,q*u,q,q,q*x*w*v))*v)/((1-x*w*v)*(1-v))
   -((Hn1(n-1,q,q*u,q*v,q,q*x*w)))/
     ((1-x*w)*(1-v)) )*w))/((1-u)*(1-w))) )
       *product('1-q^i','i'=(4*(n-1)+1)..4*n))
fi;
end:
```

Appendix D

Program with optimization for faster computation of linear extensions

```
Gn := proc(n,q,u,v,w,x) options remember;
      return simplify( Hn(n,q,u,v,w,x)+Hn(n,q,u,x,w,v))
end:

Hn := proc(n,q,u,v,w,x) options remember;
      if n=1 then
       return simplify(((1)/
         ((1-u)*(1-u*x)*(1-u*v*x)*(1-u*v*w*x)))
           *product('1-q^i','i'=1..4))
      else
       return simplify((
         (-((Gn(n-1,q,q*u,v*q*x*w,q,q))*w)/
           ((1-u)*(1-v*x*w)*(1-x)*(1-w))
         +((Gn(n-1,q,q*u,v*q*x,q*w,q)))/
           ((1-u)*(1-v*x)*(1-x)*(1-w))
```

```
      +((Gn(n-1,q,q*u*x,v*q*w,q,q))*x*w)/
        ((1-u*x)*(1-v*w)*(1-w)*(1-x))
      -((Gn(n-1,q,q*u*x,v*q,q*w,q))*x)/
        ((1-u*x)*(1-v)*(1-x)*(1-w))
      -((Gn(n-1,q,q*u*x*v*w,q,q,q))*x*(v^2)*(w^3))/
        ((1-u*x*v*w)*(1-x*v*w)*(1-v*w)*(1-w))
      +((Gn(n-1,q,q*u*x*v,q,q*w,q))*x*(v^2))/
        ((1-u*v*x)*(1-x*v)*(1-v)*(1-w)))-

      ((((Hn(n-1,q,q*u,q,q*w,v*x*q)*v)/((1-v)*(1-x*v))
      +(Hn(n-1,q,q*u,q*v*x,q*w,q))/((1-v*x)*(1-x))
      -(Hn(n-1,q,q*u,q*v,q*w,q*x))/((1-v)*(1-x)))

      -(((Hn(n-1,q,q*u,q,q,v*w*x*q)*v*w)/((1-v*w)*(1-x*v*w))
      +(Hn(n-1,q,q*u,q*v*w*x,q,q))/((1-v*w*x)*(1-x))
      -(Hn(n-1,q,q*u,q*v*w,q,q*x))/((1-v*w)*(1-x)))*w))/
        ((1-u)*(1-w)) ))
          *product('1-q^i','i'=(4*(n-1)+1)..4*n))
    fi;
end:
```