

UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Sperner Properties of the Ideals of a Boolean Lattice

Permalink

<https://escholarship.org/uc/item/8wh6f7rc>

Author

McHard, Richard W.

Publication Date

2009

Supplemental Material

<https://escholarship.org/uc/item/8wh6f7rc#supplemental>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Sperner Properties of the Ideals of a Boolean Lattice

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Richard William McHard

December 2009

Dissertation Committee:

Dr. L. H. Harper, Chairperson

Dr. Ilya Dumer

Dr. Neal Young

Copyright by
Richard William McHard
2009

The Dissertation of Richard William McHard is approved:

Committee Chairperson

University of California, Riverside

DEDICATION

To my wife, Phyllis J. Ramella:

As I set out on such an arduous task,
You have been all, and more, that I could ask.
As challenges I faced grew ever daunting
I found your strength and patience never wanting.
And thus upon reflection do I find
Your sacrifice was truly more than mine.
So know that all I've done and all I've been
Is all for you, my guiding light, my queen!

And to my advisor, Dr. L. H. Harper:

The source of everything I may have earned
Is wisdom from a master I have learned.
Your mix of lofty standards with your patience
Has seen me through my trial and tribulations.
But all that I've accomplished surely pales
Next to your strength and courage through travails!

ABSTRACT OF THE DISSERTATION

Sperner Properties of the Ideals of a Boolean Lattice

by

Richard William McHard

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2009
Dr. L. H. Harper, Chairperson

The Boolean lattice has many applications in computer science. The subsets of a Boolean lattice that are closed downward under the ordering relation are called ideals, and their study has a rich history. Problems of combinatorial optimization that involve the ideals of a Boolean lattice include combinational circuit design, bandwidth and wire length.

A graded poset P is such that every element has an unambiguous rank, or distance from the minimal element(s) of P in its Hasse diagram. The k -Sperner number of a graded poset P is the size of the largest subset of P with no more than k members that form a linear-ordered chain under the order relation of P . The k -Sperner number is a fundamental parameter of P in applications. It is easy to find if P is strongly Sperner; i.e., if, for any k , the k -Sperner number is the sum of the number of members of the k ranks that are largest in cardinality.

This is an algorithmic study of the Sperner properties of the ideals $\mathcal{J}(B)$ of a Boolean lattice B , which form a new lattice under set inclusion. $\mathcal{J}(B)$ is shown to be strongly Sperner if B has at most 6 generators. To do this, a new algorithm, the ideal exchange algorithm, was developed, proved correct, and used in conjunction with a flow algorithm to show that $\mathcal{J}(B)$ has a normal flow and is therefore strongly Sperner for the cases studied. This problem is computationally intensive (a double exponential in the number of generators) and was solved by breaking it into subcases that were run in parallel on the computer system at the University of California in Riverside.

TABLE OF CONTENTS

CHAPTER 1: STATEMENT OF PROBLEM	1
CHAPTER 2: PRELIMINARY CONCEPTS	9
CHAPTER 3: ELEMENTARY CATEGORY THEORY	26
CHAPTER 4: BACKGROUND ON SPERNER PROBLEMS	40
CHAPTER 5: FLOW ALGORITHMS, NORMALITY AND SPERNERITY	56
CHAPTER 6: THE TRADITIONAL IDEAL GENERATION ALGORITHM.....	68
CHAPTER 7: THE IDEAL EXCHANGE ALGORITHM	93
CHAPTER 8: NORMALITY TEST AND VERIFICATION PROGRAMS	113
CHAPTER 9: RESULTS	122
CHAPTER 10: FUTURE DIRECTIONS	159
BIBLIOGRAPHY	170
APPENDIX A: DEDEKIND NUMBERS AND RELATED SEQUENCES.....	173
APPENDIX B: NORMAL TEST PROGRAM	174
APPENDIX C: NORMAL TEST OUTPUT	175
APPENDIX D: INDEPENDENT VERIFICATION PROGRAM	176
APPENDIX E: INDEPENDENT VERIFICATION PROGRAM INPUT	177
APPENDIX F: INDEPENDENT VERIFICATION PROGRAM OUTPUT	178

LIST OF FIGURES

Figure 3.1. Identity and composition morphisms	27
Figure 3.2. Associative Law and Unit Law	27
Figure 3.3. Natural Transformation	28
Figure 3.4. Forgetful Functor	29
Figure 3.5. Hom-Functors	31
Figure 3.6. Product Diagram in a Category	32
Figure 3.7. Projections in a Product Category	32
Figure 3.8. Product Functor	33
Figure 3.9. Coproduct Diagram	33
Figure 3.10. Equalizer	34
Figure 3.11. Coequalizer	34
Figure 3.12. Pullback	35
Figure 3.13. Pushout	35
Figure 3.14. A Cone	36
Figure 3.15. A Morphism of Cones	36
Figure 3.16. A Cocone	36
Figure 3.17. A Morphism of Cocones	36
Figure 3.18. Exponential	38
Figure 6.1. Product Construction of B_4 from B_3	85
Figure 9.1. A Graded Poset that is Not Sperner	123
Figure 9.2. Equivalence Class of Non-Self-Dual Ideals of Rank 8 in $\mathcal{J}(B_4)$	131

LIST OF FIGURES (Continued)

Figure 9.3. Normal Flow Solution for $\mathcal{J}(B_0)$	132
Figure 9.4. Normal Flow Solution for $\mathcal{J}(B_1)$	133
Figure 9.5. Normal Flow Solution for $\mathcal{J}(B_2)$	134
Figure 9.6. Normal Flow Solution for $\mathcal{J}(B_3) / S_3$	136
Figure 9.7. Normal Flow Solution for $\mathcal{J}(B_4) / S_4$	138
Figure 9.8. Slack in the Flow Between Ranks 7 and 8 in $\mathcal{J}(B_5) / S_5$	146
Figure 9.9. Slack by Rank for $k = 5$	149
Figure 9.10. Slack by Whitney number for $k = 5$	149
Figure 9.11. Slack by Rank for $k = 6$	150
Figure 9.12. Slack by Whitney number for $k = 6$	150
Figure 9.13. Total execution time by rank for $k = 6$	151
Figure 9.14. Log execution time by rank for $k = 6$	152
Figure 9.15. Delta Time (Seconds) ($\mathcal{J}(B_6) - \mathcal{J}(B_6) / S_6$)	154
Figure 9.16. Quotient Poset and Ideal Poset Generation Time by Rank	154
Figure 9.17. Quotient Poset and Ideal Poset Flow Calculation Time by Rank	155
Figure 9.18. Retries by Rank for $k = 5$ Using Ideal Exchange Algorithm.....	155
Figure 9.19. Retries by Whitney Number for $k = 5$ Using Ideal Exchange Algorithm .	156
Figure 9.20. Retries by Rank for $k = 6$ Using Ideal Exchange Algorithm.....	157
Figure 9.21. Retries by Whitney Number for $k = 6$ Using Ideal Exchange Algorithm .	157
Figure 10.1. Intertwining Operator Commutative Diagram	161
Figure 10.2. Intertwining Operator $Stab : \mathcal{J}(B_1) \times \mathcal{J}(B_1) \rightarrow \mathcal{J}(B_2)$	163

LIST OF FIGURES (Continued)

Figure 10.3. $Stab : \mathcal{I}(B_2) \times \mathcal{I}(B_2) \rightarrow \mathcal{I}(B_3)$ is not an Intertwining Operator.....	166
--	-----

LIST OF TABLES

Table 6.1. Ideals of Rank 4 in $\mathcal{J}(B_3)$ in Lex Order	77
Table 6.2. Bit Position Order for Generating Max Ideals in $\mathcal{J}(B_3)$	78
Table 7.1. Ideals of Rank 8 in Lex Order in $\mathcal{J}(B_4)$	94
Table 9.1. Ideals of $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$, $\mathcal{J}(B_3)$, and $\mathcal{J}(B_4)$ in Lex Order.....	125
Table 9.2. A Symmetric Chain Decomposition of $\mathcal{J}(B_3)$	137
Table 9.3. A Symmetric Chain Decomposition of $\mathcal{J}(B_4)$	143
Table 9.4. Whitney Numbers for $\mathcal{J}(B_0)$ through $\mathcal{J}(B_6)$	145
Table 9.5. Execution Time for $\mathcal{J}(B_k) / S_k$ Using Ideal Exchange Algorithm	151
Table 9.6. Execution Time by Rank for $\mathcal{J}(B_6) / S_6$ and $\mathcal{J}(B_6)$	153
Table 9.7. Execution Time by Algorithm for $\mathcal{J}(B_6) / S_6$ (Middle Rank)	158
Table 10.1. Definition of $\phi = (i \cup j)0 + (i \cap j)1 : \mathcal{J}(B_1) \times \mathcal{J}(B_1) \rightarrow \mathcal{J}(B_2)$	164
Table 10.2. Definition of $\phi = (i \cup j)0 + (i \cap j)1 : \mathcal{J}(B_2) \times \mathcal{J}(B_2) \rightarrow \mathcal{J}(B_3)$	167
Table A.1. Integer Sequences for Monotone Boolean Functions	173

CHAPTER 1: STATEMENT OF PROBLEM

In [Hall 1998], Marshall Hall characterizes combinatorics as the branch of mathematics dealing with finite structures and focusing on problems of existence (construction), enumeration, and optimization. This thesis project addresses a long-standing problem of optimization: Is the lattice $\mathcal{J}(B_k)$ of ideals of a Boolean lattice B_k strongly Sperner? In the course of this project, the other facets of combinatorics were also applied to this problem.

A *poset* is a set with a partial order. An *ideal* of a poset is a sub-poset I that is “closed downward”; i.e., if $x \in I$ and $y \leq x$, then $y \in I$. The ideals $\mathcal{J}(P)$ of a poset P form a new poset under set inclusion.

A *chain* in a poset P is a totally ordered subset of P . An *antichain* in a poset P is a subset of P with no order relations between any two distinct elements. A *k-family* of a poset P is a subset of P with no chain longer than k (and so an antichain is a 1-family).

$\mathcal{J}(P)$ is *ranked* by cardinality. A ranked poset P is

- *Sperner* if the largest rank is an antichain of maximum size;
- *k-Sperner* if the k largest ranks form a k -family of maximum size;
- *strongly Sperner* if P is k -Sperner for all k .

Order-preserving functions, or *isotone* functions, are important in this study.

Given posets P and Q , a function $f: P \rightarrow Q$ is isotone iff $f(p_1) \leq f(p_2)$ whenever $p_1 \leq p_2$ for all p_1 and p_2 in P .

In a sense, ideals and antichains are “the same concept” for a given finite poset P , and in the case $P = B_k$, isotone functions represent the same concept as well. These results are “cryptomorphisms” (equivalences that are not immediately apparent and perhaps surprising at first glance) and are proven in the two propositions in this chapter.

In order to do this, we need the following two definitions.

A *maximal element* of a poset P is an element p in P such that $p \leq q$ implies $p = q$ for all q in P .

Let S be a subset of a poset P . The *downward closure* of S is the set $\downarrow_S = \{d \in P \mid d \leq s \text{ for some } s \in S\}$. $\downarrow_S$ is an ideal (see Corollary 1.2). $\downarrow_S$ is called the *ideal generated by S* .

Proposition 1.1: There is a 1-1 correspondence between the ideals of a finite poset P and the antichains of P ; that is:

- a) An antichain in P generates a unique ideal of P by downward closure.
- b) Each ideal of P has a unique antichain in P consisting of all its maximal elements.
- c) The map in a) that generates the ideal from an antichain and the map in b) that generates the antichain of maximal elements from an ideal are inverses of each other.

Proof:

- a) Let A be an antichain of P , and let $\downarrow_A$ be the downward closure of A . Suppose $x \in \downarrow_A$ and $y \leq x$. Since $x \in \downarrow_A$, $x \leq a$ for some $a \in A$. Since $y \leq x$ and $x \leq a$, $y \leq a$ by transitivity of $\leq$. Therefore $\downarrow_A$ is closed downward; i.e., $\downarrow_A$ is an ideal. In fact, $\downarrow_A$ is the smallest ideal containing A . For suppose I is an ideal containing A and suppose $d \in \downarrow_A$.

Then $d \leq a$ for some $a \in A$. Since $A \subseteq I$, $a \in I$. Since I is closed downward, $d \in I$.

Therefore $\downarrow_A \subseteq I$.

b) Since P is finite, any nonempty subset of P has maximal elements. Thus if I is an ideal, I has maximal elements. The set A of all maximal elements of I form an antichain. For suppose $a < b$ for some a and b in A . Then a is not maximal in I , a contradiction. The set of all maximal elements of an ideal is clearly unique.

c) Let A be an antichain in P and let $\downarrow_A$ be the downward closure of A . Suppose x is in $\downarrow_A$ but x is not in A . Since $x \in \downarrow_A$, $x \leq a$ for some $a \in A$. But then $x < a$ because a is in A but x is not in A , so x is not maximal in $\downarrow_A$. Now suppose y is in A . Then y is in $\downarrow_A$ because $y \leq y$. Suppose further that $y \leq z$ for some $z \in \downarrow_A$. Since $z \in \downarrow_A$, $z \leq w$ for some w in A . Then $y \leq w$, and y and w are both in A . If $y \neq w$, then $y < w$ and thus A is not an antichain. Then $y = z = w$ and so y is maximal in $\downarrow_A$. Therefore A is the antichain of maximal elements in $\downarrow_A$.

Conversely, let I be a finite ideal of P , let A be the antichain of maximal elements of I , and let $\downarrow_A$ be the downward closure of A . Suppose $x \in I$. Then $x \leq a$ for some a in A , so $x \in \downarrow_A$. Now suppose $y \in \downarrow_A$. Then $y \leq a$ for some $a \in A$. Since $A \subseteq I$, $a \in I$. Since I is an ideal, $a \in I$ and $y \leq a$, it follows that $y \in I$. Therefore $I = \downarrow_A$.

Note that the argument in a) that shows that $\downarrow_A$ is an ideal does not use the fact that A is an antichain. Thus we have

Corollary 1.2: Let P be a poset and let $S \subseteq P$. Then $\downarrow_S$ is an ideal.

Proposition 1.3: There is a 1-1 correspondence between the ideals of a finite Boolean lattice B_k and the isotone functions $f: B_k \rightarrow B_1 = \{0 < 1\}$; that is:

a) Given an ideal I of B_k , there is a unique isotone function $f: B_k \rightarrow B_1$ such that $I = f^{-1}(0) = \{x \in B_k \mid f(x) = 0\}$.

b) Conversely, given an isotone function $f: B_k \rightarrow B_1$, $f^{-1}(0)$ is an ideal.

Proof:

a) Let I be an ideal of B_k , and define $f: B_k \rightarrow B_1$ by $f(x) = 0$ if $x \in I$, $f(x) = 1$ otherwise. Then $I = f^{-1}(0)$. Suppose $x \leq y$ in B_k . If $f(x) > f(y)$, then $f(x) = 1$ and $f(y) = 0$. By construction of f , y is in I and x is not in I . But since $x \leq y$, this implies that I is not an ideal, a contradiction. Therefore $f(x) \leq f(y)$ and f is isotone. Clearly, f is the unique function $f: B_k \rightarrow B_1$ such that $I = f^{-1}(0)$.

b) Let $f: B_k \rightarrow B_1$ be an isotone function. Let $y \in f^{-1}(0)$ and let $x \leq y$. Since f is isotone, $f(x) \leq f(y)$. Since $f(y) = 0$, $f(x) = 0$, so $x \in f^{-1}(0)$. Therefore $f^{-1}(0)$ is an ideal.

The specific problem addressed herein is the conjecture that the lattice $\mathcal{I}(B_k)$ of ideals of the Boolean lattice B_k is strongly Sperner for all k . This problem is of interest for its own sake as a long-standing open problem, for its relationship to other computer science problems such as circuit design, and as an example of a general, systematic approach to problems in combinatorics.

This problem is relevant to digital circuit design, because the minimization of a combinational Boolean circuit is $\mathcal{NP}$ -complete in general but can be done in polynomial time for unate functions, which are generalized isotone Boolean functions.

The problem of counting the number of isotone Boolean functions dates back to J. W. R. Dedekind ([Dedekind 1897]). It has a long and fascinating history; see, for example, [Korshunov 2003]. Recently, L. H. Harper and Sergei Bezrukov [Harper and Bezrukov] have studied ways to estimate the number of ideals in $\mathcal{J}(P)$ using random walks on the *ideal exchange graph* (Hasse diagram) of $\mathcal{J}(P)$. For this application, it is necessary to have an upper bound on the degree of any element of $\mathcal{J}(P)$ as a vertex in the ideal exchange graph. The 2-Sperner number of P provides such an upper bound. This upper bound is sharp if P is 2-Sperner and unimodal.

The approach taken to this problem is of interest in its own right. Combinatorics has historically been a collection of loosely related results deduced by ad hoc methods. For instance, the discrete poset D_k of k unordered elements is trivially strongly Sperner, and $B_k = \mathcal{J}(D_k)$ is strongly Sperner by Sperner's Theorem [Sperner 1928]; see [Engel 1997] for details. In [Harper 2004], L. H. Harper showed that the functor $\mathcal{J} : \mathbf{Poset} \rightarrow \mathbf{Poset}$ taking each poset P to $\mathcal{J}(P)$ is fundamental in the theory of isoperimetric problems on graphs. Harper has used morphisms on the category $\mathbf{Poset}$ to solve combinatorial optimization problems involving wire length, bandwidth, and minimum-size error-correcting codes for digital circuits. In considering the theory of Sperner problems, Harper noted that all of the extensions and variations of Sperner's Theorem involve representable functors $F : \mathbf{Poset} \rightarrow \mathbf{Poset}$ such that, if P is strongly Sperner, then $F(P)$ is strongly Sperner. The functor $\mathcal{J} : \mathbf{Poset} \rightarrow \mathbf{Poset}$ is representable.

This raises the question: Under what conditions is $\mathcal{J}(P)$ strongly Sperner if P is strongly Sperner? A systematic approach to this problem is to transform (an instance of)

it to a more computationally efficient problem whose solution implies the desired result.

Such a transformation is a morphism in an appropriate category. As Harper has observed, category theory is the roadmap to morphism country. The “categorical imperative” [Mac Lane 1998], [Goguen 1991] is to

- find an appropriate morphism for the problem;
- analyze the resulting category for structural properties (limits, equivalences, etc.);
- use these properties to prove theorems about the resulting category (properties, efficiency, special cases, etc.);
- use the morphism to find an efficient representation for the problem.

Thus the categorical imperative, if successful, provides a systematic way to find better algorithms for combinatorial problems.

Chapter 2 defines mathematical concepts that are prerequisites for the remainder of the dissertation.

Chapter 3 defines the elementary concepts of category theory that are most pertinent to this study.

Chapter 4 presents the elementary concepts of $\mathcal{I}(B_k)$, the lattice of ideals of the Boolean lattice B_k , including a referenced summary of relevant existing results.

Chapter 5 discusses flow-based algorithms for determining Sperner properties of posets, including tests for strong Spernerity and normality.

Chapter 6 defines the traditional ideal generation algorithm and a modification (“selective ideal generation”) that allows multiple subcases of the normality test to be executed in parallel on adjacent ranks of $\mathcal{I}(B_k)$.

Chapter 7 defines a new ideal generation algorithm (“ideal exchange algorithm”) and proves its correctness. This new algorithm is slightly faster than the selective ideal generation algorithm on adjacent ranks, consequently improving the speed of parallel execution. The ideal exchange algorithm is used to test $\mathcal{I}(B_k)$ for normality for $k = 1, \dots, 6$ and represents the optimization aspect of combinatorics.

Chapter 8 discusses the computer program used to test for normality and an independent verification program that cross-checks the computations.

Chapter 9 lists the results of this algorithm for $k = 1, \dots, 6$ and compares the execution times of the ideal exchange algorithm and the selective ideal generation algorithm, representing the enumeration aspect of combinatorics. This chapter also gives a symmetric chain decomposition of $\mathcal{I}(B_k)$ for $k = 1, \dots, 4$ (implying that each is strongly Sperner).

Chapter 10 discusses possible directions for future research to extend the results of this research, representing the existence and construction aspects of combinatorics.

Appendix A describes sequences related to the underlying combinatorics of this problem, including the Dedekind numbers and the number of equivalence classes of $\mathcal{I}(B_k)$ produced by the action of the symmetric group on the atoms of B_k .

Appendix B, available as a separate electronic file, contains a listing of the C program used to test $\mathcal{I}(B_k)$ for normality.

Appendix C, available as a separate electronic file, contains the output files resulting from execution of the normality test program for $k = 1, \dots, 6$.

Appendix D, available as a separate electronic file, contains a listing of the independent verification program.

Appendix E, available as a separate electronic file, contains the input files to the independent verification program.

Appendix F, available as a separate electronic file, contains the output files resulting from execution of the independent verification program.

CHAPTER 2: PRELIMINARY CONCEPTS

This chapter defines basic mathematical concepts that are used throughout the remainder of this dissertation.

Set and *membership* are primitive concepts; $x \in S$ means x is a *member* or *element* of S , and $S = \{x \mid x \in S\}$. A nonempty set whose members are themselves sets with some common property is also called a *family*.

$A \subseteq B$ (A is a *subset* of B) iff $(\forall x)\{x \in A \rightarrow x \in B\}$; $B \supseteq A$ iff $A \subseteq B$.

$A = B$ iff $A \subseteq B$ and $B \subseteq A$.

$A \subset B$ (A is a *proper subset* of B) iff $A \subseteq B$ and $A \neq B$; $B \supset A$ iff $A \subset B$.

$A \cup B$ (A *union* B) = $\{x \mid x \in A \text{ or } x \in B\}$.

$A \cap B$ (A *intersect* B) = $\{x \mid x \in A \text{ and } x \in B\}$.

A' (A *complement*) = $\{x \mid x \notin A\}$.

$B \setminus A = B \cap A'$ (*set difference*), and so $A' = U \setminus A$, where U is the (problem specific) *universe* of all possible set members.

$A \oplus B = (A \setminus B) \cup (B \setminus A)$ (*symmetric difference*).

The *null set* (*empty set*) $\emptyset$ is the unique set $\emptyset$ with no members.

If $A \cap B = \emptyset$, A and B are *disjoint* and $A \cup B$ may be written $A + B$ (*disjoint union*).

The *power set* $\mathcal{P}(S)$ (also 2^S) of a set S is defined by $\mathcal{P}(S) = \{A \mid A \subseteq S\}$.

For sets S and T , $S \times T = \{(s, t) \mid s \in S \text{ and } t \in T\}$ (*Cartesian product*).

$S^2 = S \times S$ and $S^n = S^{n-1} \times S$ for $n > 2$.

The set $\mathcal{N}$ of *natural numbers* is the set of non-negative integers $\{0, 1, 2, \dots\}$.

For $0 \neq k \in \mathcal{N}$, $\mathcal{N}_k$ is the set $\{0, \dots, k-1\}$.

The number of combinations of n things taken k at a time is written $\binom{n}{k}$ (“ n choose k ”). This is the k^{th} binomial coefficient and is equal to $n! / (k!(n-k)!)$. Based on this notation, if S is a set, $\binom{S}{k}$ (“ S choose k ”) is the set of all subsets of S with exactly k members.

A *binary relation* ρ on a set U is a subset of U^2 ; $x \rho y$ means $(x, y) \in \rho$. ρ is

- 1) *reflexive* if $(\forall x \in U)\{x \rho x\}$.
- 2) *symmetric* if $(\forall x \in U)(\forall y \in U)\{(x \rho y) \rightarrow (y \rho x)\}$.
- 3) *antisymmetric* if $(\forall x \in U)(\forall y \in U)\{(x \rho y) \text{ and } (y \rho x) \rightarrow x = y\}$.
- 4) *transitive* if $(\forall x \in U)(\forall y \in U)(\forall z \in U)\{(x \rho y) \text{ and } (y \rho z) \rightarrow (x \rho z)\}$.
- 5) a *quasi-order (preorder)* if ρ is reflexive and transitive.
- 6) an *equivalence (relation)* if ρ is reflexive, symmetric and transitive.
- 7) a *partial order* if ρ is reflexive, antisymmetric and transitive.
- 8) a *total (linear) order* if ρ is a partial order and $(\forall x \in U)(\forall y \in U)\{x \rho y \text{ or } y \rho x\}$.

If ρ is a binary relation on U and $V \subseteq U$, $\rho \upharpoonright V = \rho \cap V^2$ is the *restriction* of ρ to V , and ρ is an *extension* of $\rho \upharpoonright V$ to U .

The *reflexive transitive closure* ρ^* of a relation ρ is the smallest reflexive transitive relation containing ρ .

The *inverse* ρ^{-1} of a relation ρ is defined by $\rho^{-1} = \{(y, x) \mid (x, y) \in \rho\}$.

If $\sim$ is an equivalence on U and $x \in U$, $[x] = \{y \in U \mid x \sim y\}$ is the *equivalence class* of x ; x is the *class representative* of $[x]$.

A family $\mathcal{F}$ of subsets of a universe U is *exhaustive* if every member of U is in at least one member of $\mathcal{F}$.

A *partition* of a set U is a family of nonempty, pairwise disjoint and exhaustive subsets (*blocks*) of U .

The equivalence classes of an equivalence on U form a partition of U . Conversely, a partition of U induces an equivalence on U (membership in the same block). Thus equivalence classes and partitions are “the same concept”. This is the classic instance of a cryptomorphism. (See Chapter 1 for a cryptomorphism pertinent to this study).

A subset f of $D \times C$ is *single-valued* if $(\forall x \in D)(\forall y \in C)(\forall z \in C)\{(x, y) \in f \text{ and } (x, z) \in f \rightarrow y = z\}$.

A subset f of $D \times C$ is *total* if $(\forall x \in D)(\exists y \in C)\{(x, y) \in f\}$.

A *function* (also *transformation*, *mapping*, *map*) $f : D \rightarrow C$ is a subset of $D \times C$ that is single-valued and total; D is the *domain* of f and C is the *codomain* of f . If f is single-valued but not total, f is a *partial function* from D to C .

If $f : D \rightarrow C$ is a (partial) function, $f(x) = y$ unambiguously means $(x, y) \in f$.

If $f : D \rightarrow C$ is a function and $S \subseteq D$, $f(S) = \{f(x) \mid x \in S\}$ is the *image* of S under f .

In particular, $f(D) \subseteq C$ is the *image* of f .

The function $f^{-1} : C \rightarrow \mathcal{P}(D)$ defined by $f^{-1}(y) = \{x \mid f(x) = y\}$ is the *inverse* of f . If $S \subseteq C$, $f^{-1}(S) = \bigcup_{y \in S} f^{-1}(y)$ is the *pre-image* (*inverse image*) of S under f . If f is total, the inverse images of elements of $f(D)$ form a partition of D .

The function $i_D : D \rightarrow D$ defined by $i_D(x) = x \ \forall x \in D$ is the *identity function* on D .

If $f : A \rightarrow B$ and $g : B \rightarrow C$ are functions, the *composition* $gf : A \rightarrow C$ is defined by $(gf)(x) = g(f(x)) \ \forall x \in A$.

A function $f : D \rightarrow C$ is *one-to-one* (*injective*) if $(\forall x \in D)(\forall y \in D)\{f(x) = f(y) \rightarrow x = y\}$; f is one-to-one iff f has a *left inverse* $f^l : C \rightarrow D$ such that $f^l f = i_D$.

A function $f : D \rightarrow C$ is *onto* (*surjective*) if $(\forall y \in C)(\exists x \in D)\{f(x) = y\}$; f is onto iff f has a *right inverse* $f^r : C \rightarrow D$ such that $ff^r = i_C$.

A function $f : D \rightarrow C$ is a *bijection* iff f is one-to-one and onto. A bijection defines a *1-1 correspondence* between its domain and codomain. In this case, we identify x with $\{x\}$ and thus have $f^{-1} : C \rightarrow D$, where $f^{-1} = f^l = f^r$.

A set is *finite* iff its members can be put into 1-1 correspondence with $\mathcal{N}_k$ for some non-negative integer k . A set is *denumerable* iff its members can be put into 1-1 correspondence with $\mathcal{N}$. A set is *countable* iff it is finite or denumerable; otherwise it is *uncountable*. We will be exclusively interested in countable sets, and almost always finite sets.

The set of functions from D to C is denoted C^D .

The *cardinality* $|S|$ of a finite set S is the number of members of S . Then $|\mathcal{N}_k| = k$, $|\emptyset| = 0$, $|S + T| = |S| + |T|$, $|2^S| = 2^{|S|}$, $|C^D| = |C|^{|D|}$, and $\left| \binom{S}{k} \right| = \binom{|S|}{k}$. Sets S and T (not

necessarily finite) *have the same cardinality* iff there is a 1-1 correspondence between S and T .

The *characteristic function* ϕ_S of a set S is defined by $\phi_S(x) = 1$ if $x \in S$, $\phi_S(x) = 0$ if $x \notin S$.

The *characteristic vector* of a subset S of a totally ordered countable universe $U = \{x_0 < x_1 < \dots\}$ is the string $\phi_S(x_0)\phi_S(x_1)\dots$. If U is finite, say $U = \{u_0 < u_1 < \dots < u_k\}$, it is convenient to represent ϕ_S as a string of bits in a computer. For this reason, we reverse the order of the characteristic vector to be consistent with the way bits in computer storage are ordered: most significant to least significant, left to right. All characteristic vectors in this paper will be over a finite universe and so the characteristic vectors will be understood to be in this reversed order throughout.

A *poset* $(P, \leq)$ is a set P with a partial order $\leq$ defined on P . P may refer to the poset when clear from the context. P is *discrete* if there are no relations between distinct elements of P ; i.e., $x \leq y$ iff $x = y$.

Write $y \geq x$ if $x \leq y$. The *dual* P^{op} of a poset P is the poset $(P, \geq)$.

A *weighted poset* $(P, \leq, w)$ consists of a poset $(P, \leq)$ and a map $w : P \rightarrow \mathcal{R}_+$, the set of nonnegative real numbers; w is the *weight function* of the poset. If $S \subseteq P$, $w(S) = \sum_{x \in S} w(x)$.

Given two posets $(P, \leq)$ and $(Q, \preceq)$, the *product poset* is $(P \times Q, \underline{\leq})$, where $(x, y) \underline{\leq} (z, w)$ iff $x \leq z$ and $y \preceq w$.

We may write $x < y$ or $y > x$ if $x \leq y$ and $x \neq y$.

Define $x <\bullet y$ (y covers x) if $x < y$ and $x \leq z \leq y \rightarrow x = z$ or $z = y$.

If $(P, \leq)$ is a poset and $Q \subseteq P$, $(Q, \leq_Q)$ is the *sub-poset induced by Q* , where $\leq_Q$ is the restriction of $\leq$ to Q . We typically write $(Q, \leq)$ instead of $(Q, \leq_Q)$.

A totally-ordered sub-poset is a *chain*. A chain $x_0 < x_1 < \dots < x_n$ is *saturated (skipless)* iff $x_{k-1} <\bullet x_k$ for $k = 1, \dots, n$.

If neither $x \leq y$ nor $y \leq x$, x and y are *incomparable*. A set of pairwise incomparable elements of a poset form an *antichain*. A chain and an antichain can have at most one element in common.

The *width* of a poset $(P, \leq)$ is the cardinality of its largest antichain.

If P is a poset, the *upper (lower) shadow* $\nabla(S)$ ($\Delta(S)$) of $S \subseteq P$ is the set of elements of $P \setminus S$ that cover (are covered by) at least one element of S .

A *minimal (maximal)* element m of a poset P satisfies $(\forall x \in P)\{x \leq m \rightarrow x = m\}$ $((\forall x \in P)\{m \leq x \rightarrow x = m\})$.

If an element m of a poset P satisfies $(\forall x \in P)\{m \leq x\}$ $((\forall x \in P)\{x \leq m\})$, m is a *least (greatest)* element of P . P can have at most one least (greatest) element.

Let P be a poset with least element 0 . The minimal nonzero elements of P (i.e., the minimal elements of $P \setminus \{0\}$) are called *atoms* of P . P may not have atoms in general.

A *lower (upper) bound* of a subset Q of a poset P is an element $x \in P$ such that $(\forall y \in Q)\{x \leq y\}$ $((\forall y \in Q)\{y \leq x\})$.

The *least upper bound* (*lub*, *sup*) (*greatest lower bound* (*glb*, *inf*)) of a sub-poset Q of a poset P is that element of P (unique if it exists) which is least (greatest) in the sub-poset induced by all upper (lower) bounds of Q .

If $(P, \leq)$ and $(Q, \preceq)$ are posets, $f: P \rightarrow Q$ is *isotone* (*order-preserving*, *monotone increasing*) if $(\forall x \in P)(\forall y \in P)\{x \leq y \rightarrow f(x) \preceq f(y)\}$.

Every Boolean function $f: B_k \rightarrow B_1 = \{0 < 1\}$ can be written in *sum-of-products form*; i.e., as $\bigvee_j t_j$, where each t_j is expressed as $\bigwedge_i x_i$ and each x_i is a *literal* (a Boolean variable either uncomplemented or complemented). A Boolean function $f: B_k \rightarrow B_1$ is *unate* if it can be written in sum-of-products form and no variable appears both complemented and uncomplemented in the sum-of products expression. If no variables are complemented, f is isotone.

$\preceq$ is an *extension* of $\leq$ on P iff the identity map from $(P, \leq)$ to $(P, \preceq)$ is isotone; i.e., iff $(\forall x \in P)(\forall y \in P)\{x \leq y \rightarrow x \preceq y\}$.

$(P, \leq)$ and $(Q, \preceq)$ are *isomorphic* with *isomorphism* $f: P \rightarrow Q$ if f is a bijection and both f and f^{-1} are isotone.

An *ideal* $\mathcal{I}$ (*filter* $\mathcal{F}$) of a poset P is a subset of P such that $(\forall x \in P)(\forall y \in P)(x \leq y \text{ and } y \in \mathcal{I} \rightarrow x \in \mathcal{I})$ ($(\forall x \in P)(\forall y \in P)(y \leq x \text{ and } y \in \mathcal{F} \rightarrow x \in \mathcal{F})$). An ideal (filter) with a greatest (least) element m is the *principal ideal (filter) generated by m* .

A poset P is *ranked* if there is a *rank function* $r: P \rightarrow \mathcal{N}$ such that $r(x) = 0$ for some minimal element of P and $(\forall x \in P)(\forall y \in P)\{x < \bullet y \rightarrow r(y) = r(x) + 1\}$. Denote the maximum rank of P by $r(P)$. If r has the further property that $r(m) = 0$ for each minimal

element m and all maximal elements of P have equal rank n , then P is a *graded poset of rank n* .

The i^{th} level (i^{th} rank) of a ranked poset P is $P_i = r^{-1}(i)$; $|P_i|$ is the i^{th} Whitney number W_i of P , and $w(P_i)$ is the i^{th} weighted Whitney number of P . Clearly, P_i is an antichain.

If P and Q are ranked (weighted) posets, then $P \times Q$ is a ranked (weighted) poset whose i^{th} rank is $\bigcup_{0 \leq j \leq i} P_j \times Q_{i-j}$. Furthermore, the i^{th} (weighted) Whitney number of $P \times Q$ is $\sum_{0 \leq j \leq i} W_{P_j} W_{Q_{i-j}}$, where W_{P_j} and $W_{Q_{i-j}}$ are (respectively) the j^{th} (weighted) Whitney number of P and the $i-j^{\text{th}}$ (weighted) Whitney number of Q .

A k -family of a poset is a sub-poset with no chain longer than k . A 1-family is thus an antichain.

The k -Sperner number of a poset is the cardinality of its largest k -family. Thus, the (1-)Sperner number of a poset is its width.

A ranked poset is *Sperner* if a rank with maximum Whitney number is an antichain of maximum size.

A ranked poset is k -Sperner if the union of the levels with the k largest Whitney numbers is a k -family of maximum size. A poset is *strongly Sperner* iff it is k -Sperner for all k .

A poset is *rank-unimodal* iff $W_i \geq W_{i-1}$ for all $i \leq m$ and $W_j \geq W_{j+1}$ for all $j \geq m$, where W_m is maximum.

A poset that is both rank-unimodal and strongly Sperner is a *RUSS* poset.

A poset is *rank-symmetric* iff there is a level d (typically the maximum level) such that $W_i = W_{d-i}$ for all i .

A poset that is both RUSS and rank-symmetric is a *Peck* poset.

A *graph* G is a triple $G = (V, E, \partial)$, where:

- V is a (typically finite) nonempty set of *vertices (nodes, points)*;
- E is a (typically finite) set of *edges (arcs, lines)*;
- $\partial : E \rightarrow V + \binom{V}{2}$;
- e is a *loop* around w if $\partial(e) = w$;
- $\partial(e) = \{u, v\}$ if e goes between u and v with $u \neq v$ (written $u - v$);
- if $\partial(e) = \{u, v\}$:
 - e is *incident* on u and v ;
 - u and v are *on (endpoints of) e* ;
 - u and v are *adjacent (neighbors)*.

A *directed graph (digraph)* G is a triple $G = (V, E, \partial)$, where:

- V is a (typically finite) nonempty set of *vertices (nodes, points)*;
- E is a (typically finite) set of *edges (arcs, lines)*;
- $\partial : E \rightarrow V \times V$;
- if $\partial(e) = (u, v)$, then write $u \rightarrow v$, $\partial_-(e) = u$ and $\partial_+(e) = v$; u is the *tail* and v is the *head*.

A *loop* is an edge of the form $w \rightarrow w$.

A (di)graph $G = (V, E, \partial)$ is *bipartite* if V is partitioned into *bipartitions* $M + N$ such that each edge of V is incident on (a vertex in) M and (a vertex in) N .

The *reachability* relation $\Rightarrow^*$ on a (di)graph is the reflexive transitive closure of the edge set (viewed as a binary relation).

A *path* in a (di)graph is a sequence of vertices and edges $v_0, e_1, v_1, \dots, e_k, v_k$ such that $e_j = v_{j-1} - v_j$ ($e_j = v_{j-1} \rightarrow v_j$), for $j = 1, \dots, k$; k is the *length* of the path. For a digraph, a path may be written $e_1, \dots, e_k$, where $\partial_-(e_{j+1}) = \partial_+(e_j)$ for $j = 1, \dots, k-1$. If $v_i \neq v_j$ whenever $i \neq j$, the path is *simple*.

A *cycle* in a (di)graph is a path of nonzero length from a vertex back to itself, with no vertices repeated except at the beginning and end of the path.

A (di)graph with no cycles is *acyclic*. Reachability is an equivalence on a graph and a partial order on an acyclic digraph.

A (di)graph is (*strongly*) *connected* if $x \Rightarrow^* y$ for all vertices x and y .

If $G = (V, E, \partial)$ is a (di)graph and $W \subseteq V$, the *subgraph induced by W* is (W, A, ∂) , where A is the restriction of E to $W + \binom{W}{2} (W^2)$.

The *Hasse diagram* of a poset $(P, \leq)$ is the directed graph with vertex set P and edge set $\{u \rightarrow v \mid u < \bullet v\}$. By convention, the edges are shown without arrows by writing v higher in the diagram than u if $u < \bullet v$.

Given (di)graphs G and H , a *graph (homo)morphism* $\phi : G \rightarrow H$ consists of maps $\phi_V : V_G \rightarrow V_H$ and $\phi_E : E_G \rightarrow E_H$ that preserves edges; i.e., $\partial\phi(e) = \phi\partial(e)$.

The *ideal exchange graph* of a poset P is the Hasse diagram of the ideals of P partially ordered by set inclusion (frequently viewed as an undirected graph).

A *matching* in a (di)graph $G = (V, E, \partial)$ is a set $M \subseteq E$ of non-loop edges such that no vertex in V is incident on more than one edge of M . If A and B are disjoint subsets of V , a *matching from A into B* is a matching M in the subgraph of G induced by $A + B$ such that every edge of M is incident on (a vertex in) A and (a vertex in) B , and every vertex of A is on an edge in M . The matching is a *perfect matching* if, in addition, every vertex of B is also on an edge in M , implying $|A| = |B|$.

A *tree* is an acyclic connected undirected graph, usually with a distinguished vertex r (the *root*). The relation $x \leq y$ iff there is a simple path from r to y that includes x is a partial order on the vertices of a tree.

A *network* is a digraph with distinguished *start vertex* s , distinguished *terminal vertex* t , and *capacities* $\kappa: E \rightarrow \mathcal{N}$ on the edges (alternatively, $\kappa: V \rightarrow \mathcal{N}$ on the vertices).

A *cutset* S of a network is a subset of the vertices containing the start vertex s but not the terminal vertex t . The *cut* induced by S is the set of edges connecting S and $V - S$.

A *flow* on a network is a mapping $f: E \rightarrow \mathcal{N}$ such that $f(e) \leq \kappa(e)$ for every edge e , and $\sum_{e \text{ with tail } v} f(e) = \sum_{e \text{ with head } v} f(e)$ for each vertex v except s and t (*conservation of flow*). If A and B are disjoint subsets of $V \setminus \{s, t\}$, a *flow from A to B* is a flow on the subnetwork induced by $A + B$ (technically, with new start vertex s' and new terminal vertex t' and edges $s' \rightarrow a$ for all $a \in A$ and $b \rightarrow t'$ for all $b \in B$). If A and B are disjoint subsets of $V \setminus \{s, t\}$, a *normalized flow from A to B* is a flow from A to B that sums to 1.

Given networks M and N , a *flowmorphism* $\varphi: M \rightarrow N$ is a digraph morphism onto N such that:

- $\varphi(s_M) = s_N$ and $\varphi(t_M) = t_N$

- for all $U \subseteq V_N$, $w_M(\phi^{-1}(U)) = w_N(U)$ (preservation of weights)
- for all $e \in E_N$, there is a normalized flow f_e on $\phi^{-1}(e)$ (normalized flow condition)

A ranked, positively weighted poset $(P, \leq, w)$ has the *normalized matching property (NMP)* if $w(A) / w(P_i) \leq w(\nabla A) / w(P_{i+1}) \forall A \subseteq P_i, i = 0, \dots, r(P) - 1$, where $r(P)$ is the maximum rank of P . Equivalently, P has the NMP iff P^{op} has the NMP; i.e., $w(A) / w(P_i) \leq w(\Delta A) / w(P_{i-1}) \forall A \subseteq P_i, i = 1, \dots, r(P)$.

A ranked, positively weighted poset $(P, \leq, w)$ is *normal* iff it has the NMP.

A *field of sets* is a nonempty family of sets closed under $'$ and either $\cup$ or $\cap$ (hence both). It follows that a field of sets contains the null set $\emptyset$ and the universal set U .

A *lattice* is a poset such that every pair of elements has a *sup* and an *inf*. The operators $\vee$ (*join*) and $\wedge$ (*meet*) are defined by $x \vee y = \sup(x, y)$ and $x \wedge y = \inf(x, y)$. The operators $\vee$ and $\wedge$ are *duals* of each other and satisfy the following laws:

- | | | |
|---|------------------|---|
| ○ $x \vee x = x$ | idempotence laws | $x \wedge x = x$ |
| ○ $x \vee y = y \vee x$ | commutative laws | $x \wedge y = y \wedge x$ |
| ○ $(x \vee y) \vee z = x \vee (y \vee z)$ | associative laws | $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ |
| ○ $x \vee (x \wedge y) = x$ | absorption laws | $x \wedge (x \vee y) = x$ |

Every finite lattice L has a least element $0 = \bigwedge_{x \in L} x$ and a greatest element $1 =$

$$\bigvee_{x \in L} x.$$

A ranked lattice L is *submodular* (*supermodular*) iff $r(x \vee y) + r(x \wedge y) \leq (\geq) r(x) + r(y)$. L is *modular* iff L is submodular and supermodular.

A lattice L is a *geometric lattice* iff L has a least element 0 , L is submodular, and L is generated by its atoms A ; i.e., $x = \bigvee_{a \in A, a \leq x} a$ for each $x \in L$.

A lattice is *distributive* iff the following *distributive laws* hold for all $x, y, z \in L$:

$$\circ \quad x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \quad \text{distributive laws} \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$

An element x of a lattice is *join irreducible* iff x covers exactly one element.

The ideals of a poset form a distributive lattice partially ordered by $\subseteq$; $\cup$ and $\cap$ are (respectively) the *sup* and *inf*. Conversely, every finite distributive lattice is (isomorphic to) the ideals of a poset.

A lattice L is *complemented* iff L has a least element 0 , a greatest element 1 , and for every $x \in L$ a *complement* x' such that

$$\circ \quad x \vee x' = 1 \quad \text{complement laws} \quad x \wedge x' = 0.$$

A lattice L is *Boolean* iff L is both complemented and distributive. A field of sets is a Boolean lattice partially ordered by $\subseteq$; $\cup$ and $\cap$ are (respectively) the *sup* and *inf*. Conversely, every Boolean lattice is isomorphic to a field of sets.

A *semigroup* $(S, +)$ is a set with a binary operation $+: S \times S \rightarrow S$ satisfying:

$$\circ \quad (x + y) + z = x + (y + z) \text{ for all } x, y, z \in M \quad \text{associative law}$$

A *monoid* $(M, +)$ is a semigroup with identity 0 that additionally satisfies:

$$\circ \quad x + 0 = x = 0 + x \text{ for all } x \in M \quad \text{identity law}$$

A *group* $(G, +)$ is a monoid such that every $x \in G$ has an *inverse* $-x$ satisfying:

$$\circ \quad x + -x = 0 = -x + x \quad \text{inverse law}$$

An *abelian group* $(G, +)$ is a group that additionally satisfies:

- o $x + y = y + x$ for all $x, y \in G$ commutative law

A *ring* $(R, +, \bullet)$ is a set R with binary operations $+: R \times R \rightarrow R$ and $\bullet: R \times R \rightarrow R$ satisfying:

- o $(R, +)$ is an abelian group
- o $(R, \bullet)$ is a semigroup
- o $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$ distributive laws $(y + z) \bullet x = (y \bullet x) + (z \bullet x)$

A *commutative ring* $(R, +, \bullet)$ is a ring that additionally satisfies:

- o $x \bullet y = y \bullet x$ for all $x, y \in R$ commutative law

A *ring with unit* is a ring with distinguished element 1 (*unit*) satisfying:

- o $x \bullet 1 = x = 1 \bullet x$ for all $x \in R$ identity law

A ring $(R, +, \bullet)$ is *Boolean* if it satisfies:

- o $x \bullet x = x$ for all $x \in R$ idempotence law

A Boolean ring is necessarily commutative. A Boolean ring with unit corresponds to a lattice with $x \bullet y = x \wedge y$ and $x + y = x \oplus y$ (symmetric difference or *exclusive or*); $x \vee y = x + y + x \bullet y$ and $x \oplus y = (x \wedge y') \vee (x' \wedge y)$.

A *field* $(F, +, \bullet)$ is a ring with unit $1 \neq 0$ such that $(F - \{0\}, \bullet)$ is an abelian group.

The *complex numbers* $(C, +, \bullet)$ are an *algebraically closed field*; i.e., every polynomial of degree n with complex coefficients has n (not necessarily distinct) complex roots.

A *vector space* $(V, F, +, \bullet)$ over a field $(F, +, \bullet)$ of *scalars* is a set V of *vectors* with a binary operation $+: V \times V \rightarrow V$ and a binary operation $\bullet: F \times V \rightarrow V$, satisfying:

- o $(V, +)$ is an abelian group
- o $a \bullet (\mathbf{x} + \mathbf{y}) = (a \bullet \mathbf{x}) + (a \bullet \mathbf{y})$ for all $a \in F$ and all $\mathbf{x}, \mathbf{y} \in V$
- o $(a + b) \bullet \mathbf{x} = (a \bullet \mathbf{x}) + (b \bullet \mathbf{x})$ for all $a, b \in F$ and all $\mathbf{x} \in V$
- o $a \bullet (b \bullet \mathbf{x}) = (a \bullet b) \bullet \mathbf{x}$ for all $a, b \in F$ and all $\mathbf{x} \in V$
- o $1 \bullet \mathbf{x} = \mathbf{x}$ for all $\mathbf{x} \in V$

(We abuse notation by using $+$ and $\bullet$ as operators in the vector space and in the underlying field; the operands clarify the context.)

In a vector space $(V, F, +, \bullet)$, the vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ are *linearly dependent* iff $\exists a_1, \dots, a_n \in F$, not all 0, such that

$$\sum_{i=1}^n a_i \bullet \mathbf{x}_i = 0$$

Otherwise $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ are *linearly independent*.

A *basis* for a vector space is a linearly independent set of vectors of maximum size (the *dimension* of the vector space).

When clear from the context, we use the underlying set to refer to the algebraic structure (semigroup, monoid, group, ring, field, vector space).

If X is a vector space over field F , Y is a vector space, and $\lambda : X \rightarrow Y$ satisfies $\lambda((a \bullet \mathbf{x}) + (b \bullet \mathbf{y})) = (a \bullet \lambda(\mathbf{x})) + (b \bullet \lambda(\mathbf{y}))$ for all $a, b \in F$ and all $\mathbf{x}, \mathbf{y} \in X$, λ is a *linear transformation*.

The *kernel* of a linear transformation $\lambda : X \rightarrow Y$, where X and Y are vector spaces, is defined by $\ker(\lambda) = \lambda^{-1}(0)$; λ is injective iff $\ker(\lambda) = \{0\}$.

In a ring, field, or vector space, we refer to $+$ as “addition” and $\bullet$ as “multiplication”. By convention, the order of precedence is unary operations (e.g., inverses) highest, then multiplication, then addition. We also write xy for $x\bullet y$. With these conventions, $x + yz^{-1}$ means $x + (y\bullet(z^{-1}))$.

If $f : \mathcal{R}_+ \rightarrow \mathcal{R}_+$ and $g : \mathcal{R}_+ \rightarrow \mathcal{R}_+$ are functions on the non-negative reals $\mathcal{R}_+$ and n_0, c_1 and c_2 are positive constants such that $c_1g(x) \leq f(x) \leq c_2g(x)$ for all $x \geq n_0$, f and g are the same *order of magnitude*, or $f = \Theta(g)$.

If f and g are functions on the non-negative reals and n_0 and c are positive constants such that $f(x) \leq cg(x)$ for all $x \geq n_0$, $f = O(g)$ (“ f is big Oh of g ”) and $g = \Omega(f)$.

Θ defines an equivalence on the computational complexity of functions. O defines a quasi-order on the computational complexity of functions, and thus a partial order on equivalence classes of the computational complexity of functions.

Class representatives of some equivalence classes of functions are shown below in increasing order of computational complexity:

- 1 (constants)
- $\log n$
- n
- $n \log n$
- $n^{1+\epsilon}$ for fixed ϵ such that $0 < \epsilon < 1$
- n^2
- n^k for fixed $k > 2$

- 2^n
- k^n for fixed $k > 2$

Let $\mathcal{P}$ denote the class of problems that can be solved in polynomial time. That is, a problem is in class $\mathcal{P}$ if an algorithm exists to solve each instance of it in $O(n^k)$ steps for some positive integer k , where n is the size of the input. There is an analogous class $\mathcal{NP}$ of problems that can be solved in polynomial time if the algorithm is allowed to make nondeterministic choices (equivalently, if a solution can be verified in polynomial time). Clearly, $\mathcal{P} \subseteq \mathcal{NP}$. It is conjectured as a famous open problem that the inclusion is strict (i.e., $\mathcal{P} \subset \mathcal{NP}$). A problem in $\mathcal{NP}$ that is not known to be in $\mathcal{P}$ is considered “intractable”. In particular, a problem whose only known algorithms require exponential time is considered intractable.

In analyzing the complexity of algorithms, one approximation that is particularly useful is *Stirling’s approximation*: $n! \sim (\sqrt{2\pi n}) \cdot (n/e)^n$, where $\sim$ signifies that the two formulas are *asymptotically equivalent*; i.e., their ratio goes to 1 in the limit as $n \rightarrow \infty$.

Two other functions that will be used are the *floor function* $\lfloor x \rfloor$, which is the greatest integer less than or equal to x , and the *ceiling function* $\lceil x \rceil$, which is the smallest integer greater than or equal to x .

CHAPTER 3: ELEMENTARY CATEGORY THEORY

The concepts of *function* and *structure* occur throughout mathematics. The branch of mathematics that unifies these concepts is *category theory*, which has been called “the algebra of functions”. [Mac Lane 1998] is the classic reference in the field. [Awodey] is a very readable reference. [Rydeheard and Burstall 1988] is a computational view of category theory.

A (small) category $\mathbf{C}$ consists of a set O (objects), a set A (morphisms or arrows), and the following four maps:

- o $dom : A \rightarrow O$; $dom(f)$ is the *domain* of morphism f
- o $cod : A \rightarrow O$; $cod(f)$ is the *codomain* of morphism f
- o $id : O \rightarrow A$ such that $dom(1_x) = x = cod(1_x)$ for all objects $x \in O$, where $1_x = id(x)$
- o *composition* $(\bullet)$, a partial function $\bullet : A \times A \rightarrow A$, defined for (g, f) precisely when $dom(g) = cod(f)$ and written $g \bullet f$ or gf , such that $dom(gf) = dom(f)$ and $cod(gf) = cod(g)$. Finally, $\bullet$ satisfies the following two laws whenever $\bullet$ is defined:
 - o $(h \bullet g) \bullet f = h \bullet (g \bullet f)$ *associative law*
 - o $1_b \bullet f = f$ and $g \bullet 1_b = g$ *unit law*

A morphism $f : d \rightarrow c$ that has an *inverse* $g : c \rightarrow d$ such that $fg = 1_c$ and $gf = 1_d$ is an *isomorphism*. An isomorphism is unique if it exists.

A category has an underlying *diagram graph*, a digraph showing the morphisms between objects, built from the basic building blocks shown in Figure 3.1. Conversely, a

digraph G can be viewed as the underlying graph of a category whose arrows are the paths of G .

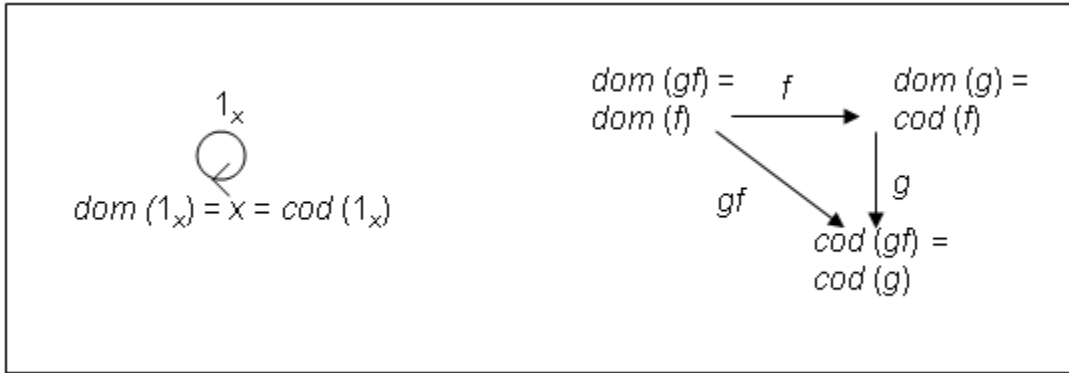


Figure 3.1. Identity and composition morphisms

In Figure 3.1, the two paths from $\text{dom}(f)$ to $\text{cod}(g)$ are equivalent; i.e., the *diagram commutes*. Figure 3.2 shows commutative diagrams for the associative law and unit law.

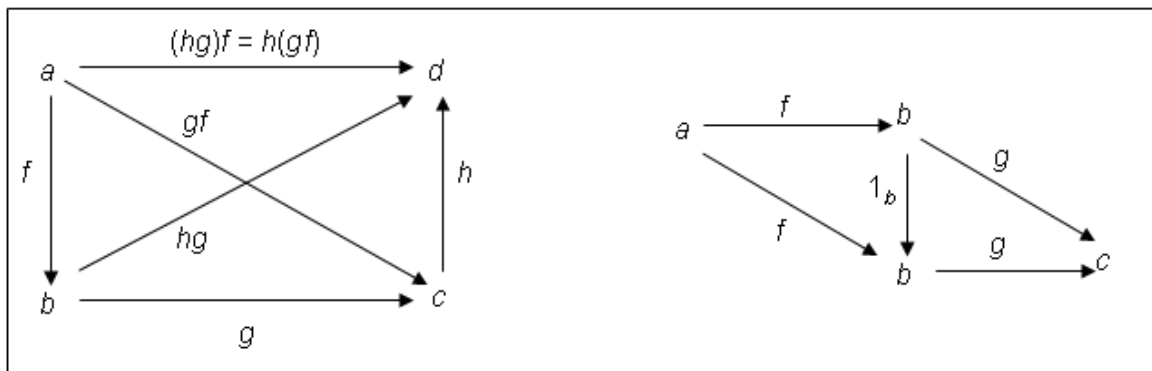


Figure 3.2. Associative Law and Unit Law

Given categories $\mathbf{C}$ and $\mathbf{B}$, a (covariant) *functor* $T : \mathbf{C} \rightarrow \mathbf{B}$ consists of an *object function* $T_O : O_{\mathbf{C}} \rightarrow O_{\mathbf{B}}$ and a *morphism function* $T_A : A_{\mathbf{C}} \rightarrow A_{\mathbf{B}}$ such that:

- o If $f : d \rightarrow c$, then $Tf : Td \rightarrow Tc$
- o $T_{1c} = 1_{Tc}$
- o $T(g \bullet f) = Tg \bullet Tf$

Thus a functor is a morphism in the category **Cat** of categories. (Note that we write T for T_O and T_A when clear from the context.)

Given functors $S : \mathbf{C} \rightarrow \mathbf{B}$ and $T : \mathbf{C} \rightarrow \mathbf{B}$, a *natural transformation* $\tau : S \rightarrow T$ is a family of morphisms $\{\tau d \mid d \in O_{\mathbf{C}}\}$ in $\mathbf{B}$, one for each object in $\mathbf{C}$, such that the diagram in Figure 3.3 commutes for each morphism $f : d \rightarrow c$ of $\mathbf{C}$. The morphism τd is the *component* of τ at d . If each component of τ is an isomorphism, τ is a *natural isomorphism*.

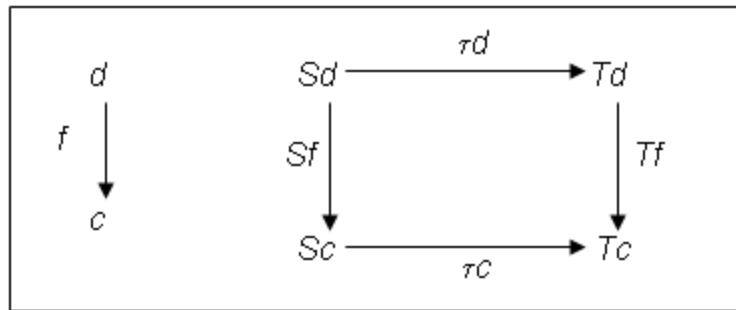


Figure 3.3. Natural Transformation

A natural transformation is a morphism of functors. Given categories $\mathbf{C}$ and $\mathbf{B}$, the *functor category* $\text{Fun}(\mathbf{C}, \mathbf{B})$ has:

- o objects: functors $S : \mathbf{C} \rightarrow \mathbf{B}$

- o morphisms: natural transformations $\tau : S \rightarrow T$ for functors $S, T : \mathbf{C} \rightarrow \mathbf{B}$
- o for each functor S , $1_S : S \rightarrow S$ is the identity natural transformation
- o $(\tau \bullet \sigma)d = (\tau d) \bullet (\sigma d)$

One example of a functor is $F : \mathbf{Cat} \rightarrow \mathbf{Graph}$ that takes a category and maps it to its underlying diagram graph (see Figure 3.4). F is called a *forgetful functor* because F “forgets” the extra structure in $\mathbf{Cat}$ that is not in $\mathbf{Graph}$. $\mathbf{Set}^{\mathbf{Graph}}$ is a functor category.

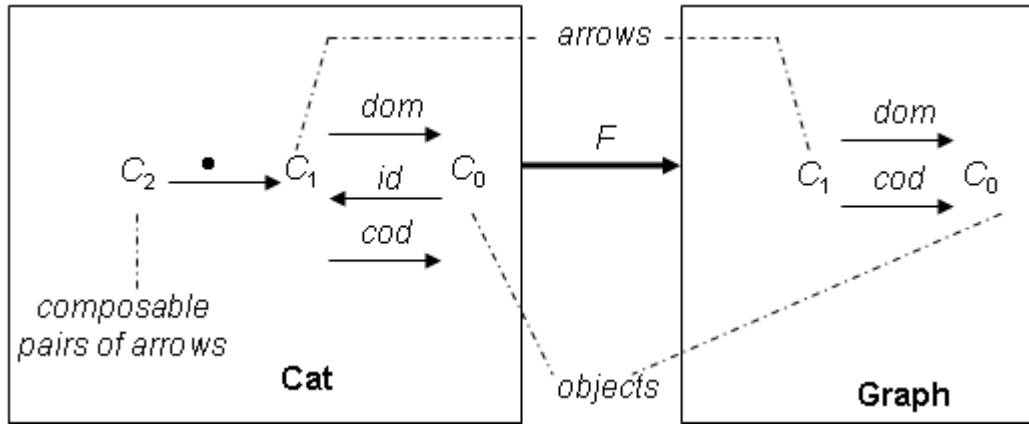


Figure 3.4. Forgetful Functor

The *dual category* (opposite category) $\mathbf{C}^{\text{op}}$ of a category $\mathbf{C}$ is the category obtained by reversing all the morphisms; i.e., $\text{dom}(f^{\text{op}}) = \text{cod}(f)$ and $\text{cod}(f^{\text{op}}) = \text{dom}(f)$. A (covariant) functor $S : \mathbf{C}^{\text{op}} \rightarrow \mathbf{B}$ induces a *contravariant functor* $S^{\text{op}} : \mathbf{C} \rightarrow \mathbf{B}$, such that $S^{\text{op}}f = Sf^{\text{op}}$. Note that $S^{\text{op}}(f \bullet g) = S^{\text{op}}g \bullet S^{\text{op}}f$.

A “sentence” S in a category $\mathbf{C}$ has a *dual sentence* S^{op} in $\mathbf{C}^{\text{op}}$ made by interchanging each *dom* and *cod* and reversing the order of composition. Note that S is true iff S^{op} is true; this is the *principle of duality*.

An important category is **Set** (the category of sets). The morphisms of **Set** are functions between sets. Note that **Set** is not itself a set in general, because there are too many sets. However, **Set** (and other categories of interest such as **Poset** and **Graph** on structured sets) are *locally small* in the following sense: for fixed domain d and codomain c , the morphisms from d to c are collectively (small enough to be) a set. This set, called a *hom-set*, is written $\text{Hom}(d, c) = \{f \mid \text{dom}(f) = d \text{ and } \text{cod}(f) = c\}$. We will be exclusively interested in locally small categories. Morphisms f, g in $\text{Hom}(d, c)$ for fixed d and c are *parallel arrows*.

If $\mathbf{C}$ is locally small, for each fixed object $a \in \mathbf{C}$ there is an induced (covariant) *hom-functor* $\text{Hom}(a, -) : \mathbf{C} \rightarrow \mathbf{Set}$ that takes each object $b \in \mathbf{C}$ to $\text{Hom}(a, b)$ and each morphism $h : b \rightarrow x$ to the function $h_* : \text{Hom}(a, b) \rightarrow \text{Hom}(a, x)$ defined by $h_*f \rightarrow hf$ for each $f \in \text{Hom}(a, b)$ (“composition on the left”). A functor is *representable* if it is (naturally isomorphic to) a hom-functor $\text{Hom}(a, -)$; a is the *representing object*.

The induced *contravariant hom-functor* is $\text{Hom}(-, b) : \mathbf{C}^{\text{op}} \rightarrow \mathbf{Set}$ that takes each object $a \in \mathbf{C}$ to $\text{Hom}(a, b)$ and each morphism $g : a \rightarrow y$ to $g^* : \text{Hom}(y, b) \rightarrow \text{Hom}(a, b)$ defined by $g^*f \rightarrow fg$ for each $f \in \text{Hom}(y, b)$ (“composition on the right”).

Figure 3.5 is the commutative diagram for covariant and contravariant hom-functors.

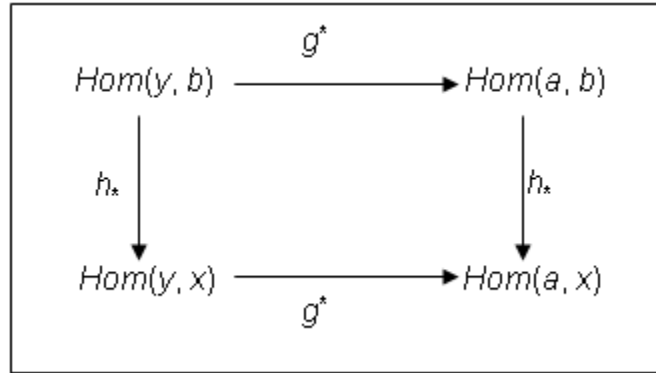


Figure 3.5. Hom-Functors

Some categories contain their own hom-sets and hom-functors. Examples are **Set**, **Vect**, and **Poset** (with functions ordered pointwise). In these cases the hom-set and induced hom-functor are *internal*, and we write $hom(a, b)$ instead of $Hom(a, b)$.

An object s in a category is *initial* iff, for every object a , there is a unique morphism $s \rightarrow a$. An object t in a category is *terminal* iff, for every object a , there is a unique morphism $a \rightarrow t$.

A *product (diagram)* in a category **C** is an object P with morphisms $\pi_1 : P \rightarrow A$ and $\pi_2 : P \rightarrow B$ with the property that, for all $x_1 : X \rightarrow A$ and $x_2 : X \rightarrow B$, there is a unique $!u : X \rightarrow P$ such that the diagram in Figure 3.6 commutes. Write $P = A \times B$. A diagram with a unique morphism such as u is said to have a *universal mapping property (UMP)*.

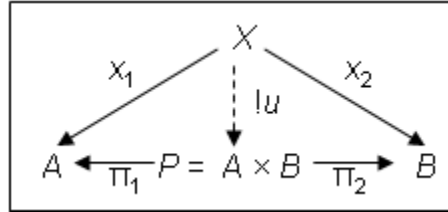


Figure 3.6. Product Diagram in a Category

A category $\mathbf{C}$ has (*finite*) *products* if $\mathbf{C}$ has a terminal object (0-ary product) and every pair of objects has a product diagram. (The objects of $\mathbf{C}$ are unary products.)

The category **Cat** of categories has products. Given categories $\mathbf{B}$ and $\mathbf{C}$, the *product* $\mathbf{B} \times \mathbf{C}$ is the category with objects and morphisms defined as the usual Cartesian products with $(f, g) \bullet (h, k) = (f \bullet h, g \bullet k)$. The *projections* $P : \mathbf{B} \times \mathbf{C} \rightarrow \mathbf{B}$ and $Q : \mathbf{B} \times \mathbf{C} \rightarrow \mathbf{C}$ are defined by $P(b, c) = b$, $P(f, g) = f$, $Q(b, c) = c$, $Q(f, g) = g$ for objects $b \in \mathbf{B}$ and $c \in \mathbf{C}$ and morphisms $f \in \mathbf{B}$ and $g \in \mathbf{C}$. P and Q are *universal* functors. That is, given any category $\mathbf{D}$ with functors $S : \mathbf{D} \rightarrow \mathbf{B}$ and $T : \mathbf{D} \rightarrow \mathbf{C}$, S and T “factor through” P and Q via a unique functor $!F$ as shown in the UMP of Figure 3.7.

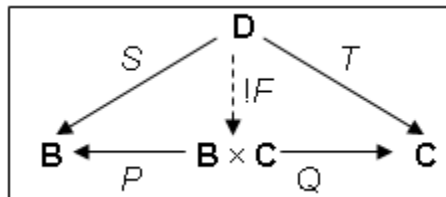


Figure 3.7. Projections in a Product Category

The *product* $U \times V : \mathbf{B} \times \mathbf{C} \rightarrow \mathbf{B}' \times \mathbf{C}'$ of functors $U : \mathbf{B} \rightarrow \mathbf{B}'$ and $V : \mathbf{C} \rightarrow \mathbf{C}'$ is defined by $(U \times V)(b, c) = (Ub, Vc)$ and $(U \times V)(f, g) = (Uf, Vg)$. It is the unique (!) functor such that the diagram in Figure 3.8 commutes.

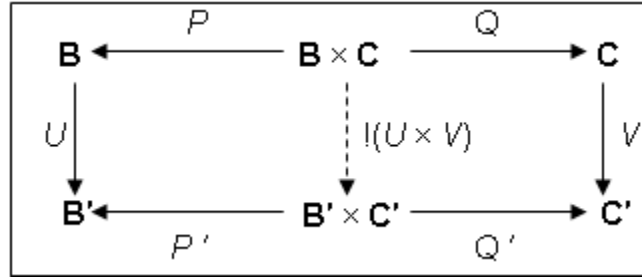


Figure 3.8. Product Functor

The dual of product is *coproduct*. Its UMP is shown in Figure 3.9.

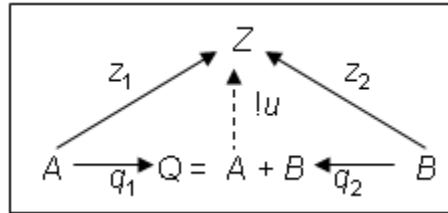


Figure 3.9. Coproduct Diagram

Given parallel arrows f, g in $\text{Hom}(A, B)$, an *equalizer* of f and g is an object E and a morphism $e : E \rightarrow A$ such that $fe = ge$ with the following UMP: for any object Z and morphism $z : Z \rightarrow A$, there is a unique $!u : Z \rightarrow E$ such that the diagram in Figure 3.10 commutes.

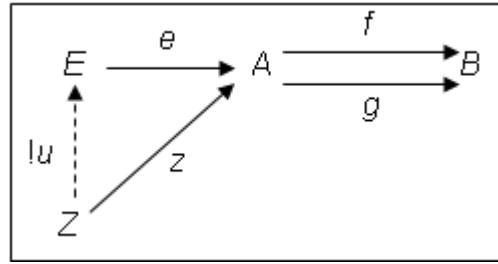


Figure 3.10. Equalizer

The dual of equalizer is *coequalizer*. The UMP is shown in Figure 3.11.

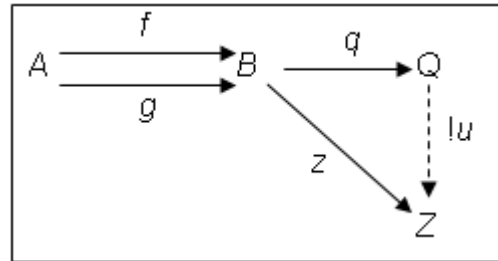


Figure 3.11. Coequalizer

Given morphisms $f: A \rightarrow C$ and $g: B \rightarrow C$ with common codomain, a *pullback* (unique up to isomorphism) of f and g is a pair of morphisms $p_1: P \rightarrow A$ and $p_2: P \rightarrow B$ such that every pair of morphisms $z_1: Z \rightarrow A$ and $z_2: Z \rightarrow B$ uniquely factor through a morphism $!u: Z \rightarrow P$ as shown in the UMP of Figure 3.12.

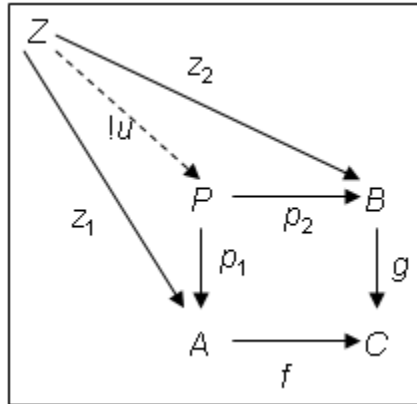


Figure 3.12. Pullback

The dual of a pullback is a *pushout*. Its UMP is shown in Figure 3.13.

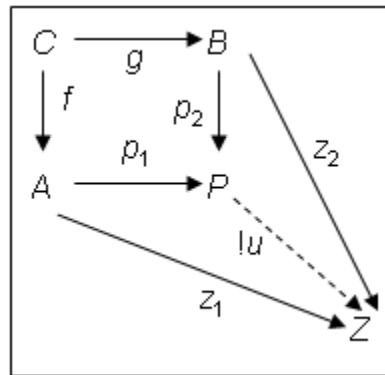


Figure 3.13. Pushout

A functor $D : \mathbf{J} \rightarrow \mathbf{C}$ is a “diagram of $\mathbf{J}$ in $\mathbf{C}$ ” viewing $\mathbf{J}$ as an index set and writing $D(j) = D_j$ for each object of $\mathbf{J}$. A *cone* to a diagram D (the *base*) is an object C in $\mathbf{C}$ and a family of morphisms $\{c_j : C \rightarrow D_j\}$ in $\mathbf{C}$, one morphism for each j in $\mathbf{J}$, such that for each morphism $m : i \rightarrow j$ in $\mathbf{J}$, the triangle in Figure 3.14 commutes. A *morphism of*

cones is a morphism $f : C \rightarrow C'$ in $\mathbf{C}$ such that, for each j in $\mathbf{J}$, the triangle in Figure 3.15 commutes.

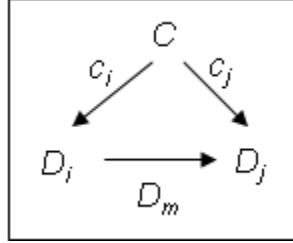


Figure 3.14. A Cone

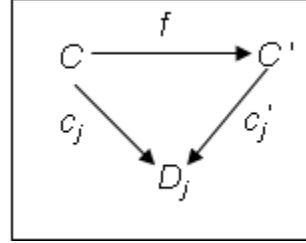


Figure 3.15. A Morphism of Cones

$\mathbf{Cone}(D)$ is thus a category. A *limit* for a diagram $D : \mathbf{J} \rightarrow \mathbf{C}$ is a terminal object in $\mathbf{Cone}(D)$, written $p_i : \lim_{\leftarrow j} D_j \rightarrow D_i$. The limit is *finite* if $\mathbf{J}$ is finite. The UMP is: given cone $(C, \{c_j\})$ to D , $\exists! u_C : C \rightarrow \lim_{\leftarrow j} D_j$ such that $p_j u_C = c_j$ for all j in $\mathbf{J}$. A category $\mathbf{C}$ has all finite limits iff every finite diagram $D : \mathbf{J} \rightarrow \mathbf{C}$ has a limit in $\mathbf{C}$.

The dual concept of limit is *colimit*, written $p_i : D_i \rightarrow \lim_{\rightarrow j} D_j$, a terminal object in $\mathbf{Cocone}(D)$. The UMP for cocone $(C, \{c_j\})$ is $\exists! u_C : C \rightarrow \lim_{\rightarrow j} D_j$ such that $u_C c_j = p_j$ for all j in $\mathbf{J}$. Figures 3.16 and 3.17 for cocones are analogous to Figures 3.14 and 3.15 for cones.

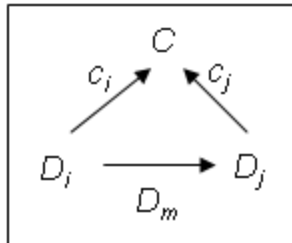


Figure 3.16. A Cocone

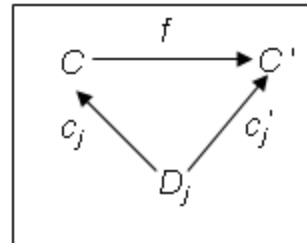


Figure 3.17. A Morphism of Cocones

A significant fact is that representable functors preserve (co)limits. A pullback (pushout) is a special case of a (co)limit. Products, equalizers, pullbacks and limits are more generally related; given category $\mathbf{C}$, the following statements are equivalent:

- $\mathbf{C}$ has all finite (co)limits.
- $\mathbf{C}$ has all finite (co)products and (co)equalizers.
- $\mathbf{C}$ has pullbacks (pushouts) and a terminal (initial) object.

The equivalence proofs are all constructive. Many “concrete” categories (whose objects are sets with perhaps some structure such as graphs or posets) have coproducts and coequalizers (which are essentially quotients by equivalence classes in **Set**). Thus:

- *Concrete categories typically have pushouts.*
- *There is an algorithm to find them based on coproducts and coequalizers (i.e., disjoint unions and quotient maps).*

Let $\mathbf{C}$ have all binary products. An *exponential* of objects B and C is an object C^B together with a morphism $eval : C^B \times B \rightarrow C$ (*evaluation*) such that, given object A and morphism $f : A \times B \rightarrow C$, $\exists! f^T : A \rightarrow C^B$ (*transpose* of f) such that $eval \bullet (f^T \times 1_B) = f$. Furthermore, given $g : A \rightarrow C^B$, let $g^t = eval \bullet (g \times 1_B) : A \times B \rightarrow C$. Then $f^{Tt} = f$ and $g^{tT} = g$, so we say f and g are *transposes* of each other. The UMP for exponentials is shown in Figure 3.18.

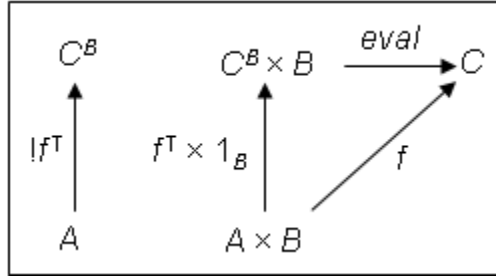


Figure 3.18. Exponential

A category that has all finite products and exponentials is a *Cartesian closed category* (CCC). Exponentiation by a fixed object in a CCC is *functorial*; i.e., $-^A : \mathbf{C} \rightarrow \mathbf{C}$ is a functor. It is worth noting that the category **Poset** of posets and isotone functions is a CCC.

An *adjunction* between categories $\mathbf{C}$ and $\mathbf{D}$ is a pair of functors $L : \mathbf{C} \rightarrow \mathbf{D}$ (*left adjoint*) and $R : \mathbf{D} \rightarrow \mathbf{C}$ (*right adjoint*) and a natural isomorphism $\varphi : \text{hom}_{\mathbf{D}}(LC, D) \approx \text{hom}_{\mathbf{C}}(C, RD) : \psi$. Right adjoints preserve limits and left adjoints preserve colimits.

Some familiar categories are **Group**, **Ring**, **Field**, **Vect**, **Graph**, and **Poset**, and their finite variations (e.g., **Poset_{fin}**). These are concrete categories; the morphisms preserve all operators and relations (i.e., $f(a * b) = f(a) * f(b)$) between objects. The restrictions of these categories to finite (structured) sets are also categories (the ones pertinent to this study). In these categories, the morphisms are called *homomorphisms*. A homomorphism necessarily preserves identities and inverses. The isomorphisms are the bijective homomorphisms whose inverses are homomorphisms.

It is worth noting that a specific poset $(P, \leq)$ is also a category $\mathbf{P}$ with objects P and morphisms $\{x \rightarrow y \mid x \leq y\}$. There cannot be more than one morphism between any two objects in $\mathbf{P}$, and the underlying diagram graph is a Hasse diagram.

CHAPTER 4: BACKGROUND ON SPERNER PROBLEMS

Several points that are fundamental to this study are defined in this chapter.

The Boolean lattice B_k can be viewed in the following equivalent ways:

- the power set of a set of k elements;
- a lattice (complemented and distributive) with k atoms that serve as generators;
- an idempotent ring with k generators;
- the product of k $\{0 < 1\}$ chains;
- the ideals (ordered, as is customary, by set inclusion) of the discrete poset D_k .

We will frequently represent an ideal I of a poset P by its characteristic vector.

Let us observe some properties of $\mathcal{J}(B_k)$. The members of B_k are subsets of an unordered k -set, so there are 2^k elements of B_k . $\mathcal{J}(B_k)$, in turn, consists of ideals (which are subsets) of B_k . Thus, 2^{2^k} is an upper bound on $|\mathcal{J}(B_k)|$.

The following elementary result gives an asymptotic lower bound on $|\mathcal{J}(B_k)|$.

Proposition 4.1: $|\mathcal{J}(B_k)|$ is asymptotically $\Omega(2^{2^{k/\sqrt{k}}})$.

Proof: Observe that ranks $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ (which may be the same rank) of B_k have $k!/((\lfloor k/2 \rfloor!) \bullet (\lceil k/2 \rceil!))$ elements. Furthermore, these are the largest ranks of B_k .

Stirling's formula approximates this as $(\sqrt{2\pi k}) \bullet (k/e)^k / (\sqrt{\pi k}) \bullet (k/2e)^{k/2}$, which simplifies

to $(\sqrt{2/\pi}) \bullet (2^{k/\sqrt{k}})$. Now each subset of rank $\lfloor k/2 \rfloor$ is a unique antichain and thus

generates a unique ideal. There are asymptotically $\Theta(2^{2^{k/\sqrt{k}}})$ such ideals, not quite a

double exponential in k . But note that this estimate only includes ideals that are

generated by a subset of the maximum-size rank, so $|\mathcal{J}(B_k)|$ is asymptotically $\Omega(2^{2^{k/\sqrt{k}}})$.

[Kleitman 1969] showed that $\log_2 |\mathcal{J}(B_k)| = O(2^k/\sqrt{k})$.

We next prove some basic properties of the poset family $\{\mathcal{J}(B_k)\}$.

Lemma 4.2: If P is a poset, $I \in \mathcal{J}(P)$, $x \in P \setminus I$ and $y \in I$, then

- a) $I + \{x\}$ is an ideal of P iff x is minimal in $P \setminus I$, and
- b) $I \setminus \{y\}$ is an ideal of P iff y is maximal in I .

Proof:

- a) Let x be minimal in $P \setminus I$, let $p \in I + \{x\}$, and let $q \leq p$. If $q \in I$, $q \in I + \{x\}$.

Suppose $q \in P \setminus I$. Then $p \in P \setminus I$ because I is an ideal. Then $p = x$. Since x is minimal in $P \setminus I$ and $q \leq x$ in $P \setminus I$, $q = x$ and so $q \in I + \{x\}$. Therefore $I + \{x\}$ is an ideal. Now suppose x is not minimal in $P \setminus I$. Then there exists $z < x$ in $P \setminus I$ and z is not in $I + \{x\}$. Therefore $I + \{x\}$ is not an ideal.

b) Let y be maximal in I , let $a \in I \setminus \{y\}$, and suppose $b \leq a$. Then $a \in I$ and (because I is an ideal) $b \in I$. If $b = y$, then $y \leq a$ and (because y is maximal in I) $y = a$. But this cannot be because $a \in I \setminus \{y\}$, so $b \neq y$. Then $b \in I \setminus \{y\}$. Therefore $I \setminus \{y\}$ is an ideal. Now suppose y is not maximal in I . Then there exists $w > y$ in I and $w \in I \setminus \{y\}$. Therefore, since $y \notin I \setminus \{y\}$, $I \setminus \{y\}$ is not an ideal.

Proposition 4.3: The minimum size ideal in $\mathcal{J}(B_k)$ is 0 and the maximum size ideal in $\mathcal{J}(B_k)$ is 2^k .

Proof: B_k is the largest ideal in $\mathcal{J}(B_k)$ and has 2^k members; $\emptyset$ is the smallest ideal of any given poset and has cardinality 0.

Proposition 4.4: $\mathcal{J}(B_k)$ is a graded poset, and the rank of an ideal in $\mathcal{J}(B_k)$ is its cardinality.

Proof: By simple induction on the size of an ideal in $\mathcal{J}(B_k)$.

Basis ($r = 0$): $\emptyset$ is an ideal with cardinality 0 and $\emptyset$ does not cover any ideal.

Induction: Let $0 \leq r < k$ and suppose each ideal of size r covers only ideals of size $r - 1$. Let I be an ideal of size $r + 1$, and let $J \subset I$ be an ideal. Let m be maximal in $I \setminus J$, and suppose $m < p$ in I . Since m is not in J and J is an ideal, p is not in J , and so $m < p$ in $I \setminus J$. But this contradicts the maximality of m in $I \setminus J$. Thus m is maximal in I . By Lemma 4.2, $I \setminus \{m\}$ is an ideal and $J \subseteq I \setminus \{m\} < \bullet I$. Therefore I covers J iff $J = I \setminus \{m\}$; i.e., iff $|J| = r$, and the induction is extended.

In fact, one can easily define an ideal of each rank from 0 through 2^k as follows:

- The null ideal has rank (cardinality) 0.
- Number the elements of B_k across by rank in B_k , then up by increasing rank in B_k . For example, the elements of B_3 are numbered as follows:

0	000
1	001
2	010
3	100
4	011
5	101
6	110
7	111

- Then $\{0, \dots, i - 1\}$ is an ideal with rank (cardinality) i . In particular, $\{0, \dots, 2^k - 1\} = B_k$ has rank (cardinality) 2^k .

We now show that the lattice of ideals of any finite poset is distributive and is generated by the sub-poset of its join irreducible elements (which is isomorphic to the underlying poset).

Proposition 4.5: Let P be a poset with $x \in P$ and $y \in P$. Let $\downarrow_x = \{z \mid z \leq x\}$ and $\downarrow_y = \{w \mid w \leq y\}$ be the ideals generated by x and y , respectively. Then $x \leq y$ in P iff $\downarrow_x \subseteq \downarrow_y$ (i.e., $\downarrow_x \subseteq \downarrow_y$) in $\mathcal{I}(P)$.

Proof: Let $x \leq y$ in P and let $z \in \downarrow_x$. By the definition of $\downarrow_x$, $z \leq x$. Since $x \leq y$ and $z \leq x$, $z \leq y$, and so $z \in \downarrow_y$ by the definition of $\downarrow_y$. Therefore $\downarrow_x \subseteq \downarrow_y$ in $\mathcal{I}(P)$.

Conversely, suppose $\downarrow_x \subseteq \downarrow_y$ in $\mathcal{I}(P)$. Since $x \in \downarrow_x$, $x \in \downarrow_y$. By the definition of $\downarrow_y$, $x \leq y$.

Proposition 4.6:

- a) The ideals $\mathcal{I}(P)$ of a poset P form a distributive lattice.
- b) If P is finite, the sub-poset J of $\mathcal{I}(P)$ induced by the join irreducible elements of $\mathcal{I}(P)$ is isomorphic to P (and thus $\mathcal{I}(J)$ is isomorphic to $\mathcal{I}(P)$).

Proof:

a) Let i and j be ideals of P , let $x \in i \cup j$, and let $y \leq x$. If $x \in i$, then $y \in i$ since i is an ideal; similarly $y \in j$ if $x \in j$. In either case, $y \in i \cup j$. Therefore $i \cup j$ is an ideal.

Let i and j be ideals of P , let $x \in i \cap j$, and let $y \leq x$. Then $x \in i$ and $x \in j$. Since i and j are ideals, $y \in i$ and $y \in j$. Then $y \in i \cap j$. Therefore $i \cap j$ is an ideal.

This shows that $\mathcal{I}(P)$ is closed under $\cap$ and $\cup$, so $\mathcal{I}(P)$ is a lattice with meet $\cap$ and join $\cup$. It is a basic property of set theory that $\cap$ and $\cup$ distribute over each other.

b) By Lemma 4.2, the immediate predecessors of an ideal i are obtained by deleting a maximal element of i . Then i is join irreducible iff i has one immediate predecessor iff i has one maximal element m iff i is a principal ideal generated by m . Then the sub-poset J of $\mathcal{J}(P)$ induced by the join irreducible elements of $\mathcal{J}(P)$ has as its elements the principal ideals generated by each of the elements of P . By Proposition 4.5, this sub-poset is isomorphic to P .

[Birkhoff 1967] proves the more general result that any finite distributive lattice L is isomorphic to the lattice of ideals of the sub-poset of join irreducible elements of L .

Next, we show one way to define the ideals in $\mathcal{J}(B_{k+1})$ in terms of the ideals in $\mathcal{J}(B_k)$. By Proposition 4.6, $\mathcal{J}(P)$ is closed under union and intersection. Define $\phi(i, j) = (i \cup j)0 + (i \cap j)1$ for ideals i and j in $\mathcal{J}(B_k)$, where $(i \cup j)0 + (i \cap j)1$ is the disjoint union of the ideal $i \cup j$ with 0 in the new coordinate position, and the ideal $i \cap j$ with 1 in the new coordinate position. See Chapter 10 for examples.

Proposition 4.7: The map $\phi : \mathcal{J}(B_k) \times \mathcal{J}(B_k) \rightarrow \mathcal{J}(B_{k+1})$ has the following properties:

- a) $\phi(i, j)$ is an ideal for all i and j .
- b) ϕ is onto.
- c) ϕ is rank-preserving; i.e., $r(\phi(i, j)) = r(i) + r(j)$.

Proof:

a) Let $y \in \phi(i, j)$ and $x < y$. If the new coordinate position of y is 0, then the new coordinate position of x is 0. Then $x = z0$, $y = w0$, $z < w$, and w is in $i \cup j$. But then

(since $i \cup j$ is an ideal) z is in $i \cup j$, so x is in $(i \cup j)0 \subseteq \phi(i, j)$. By a similar argument, if the new coordinate position of x is 1, then the new coordinate position of y is also 1 and (because $i \cap j$ is an ideal) x and y are both in $(i \cap j)1 \subseteq \phi(i, j)$. The only other possibility is that the new coordinate position of x is 0, say $x = u0$, and the new coordinate position of y is 1, say $y = v1$. Then y is in $(i \cap j)1$. Since $x < y$, $u \leq v$. Since v is in $i \cap j$ and $i \cap j$ is an ideal, u is also in $i \cap j$, hence in $i \cup j$, so x is in $(i \cup j)0 \subseteq \phi(i, j)$.

b) Let p be an ideal in $\mathcal{J}(B_{k+1})$. Let p_0 be the subset of p with the new coordinate position 0, and let p_1 be the subset of p with the new coordinate position 1. Define $p \mid 0$ to be the set consisting of the first k coordinates of elements of p_0 , and $p \mid 1$ to be the set consisting of the first k coordinates of elements of p_1 . Clearly, $p \mid 0$ and $p \mid 1$ are ideals in $\mathcal{J}(B_k)$. (For $b \in p \mid 0 \Rightarrow b0 \in p$; if $a < b$, then $a0 \in p$, so $a \in p \mid 0$. Similarly for $p \mid 1$.) Let u be in $p \mid 1$; then $x = u1$ is in p_1 , hence in p . Since p is an ideal, $y = u0$ is in p , hence in p_0 . But then u is in $p \mid 0$. It follows that $p \mid 1 \subseteq p \mid 0$, so $p = p_0 + p_1 = (p \mid 0)0 + (p \mid 1)1 = (p \mid 0 \cup p \mid 1)0 + (p \mid 0 \cap p \mid 1)1 = \phi(p \mid 0, p \mid 1)$.

c) Since ϕ is the disjoint union of $(i \cup j)0$ and $(i \cap j)1$, $|\phi(i, j)| = |i \cup j| + |i \cap j|$.

By the Principle of Inclusion and Exclusion, $|i \cup j| + |i \cap j| = |i| + |j|$.

Corollary 4.8: $|\mathcal{J}(B_{k+1})| \leq |\mathcal{J}(B_k)|^2$.

Proof: Follows immediately from b) of Proposition 4.7.

It is shown next that a recurrence defined by $f(k+1) = f(k)^2$ has a double exponential solution.

Proposition 4.9: Let $S(k)$ be defined by the recurrence

- $S(0) = x$
- $S(k) = S(k-1)^2$

Then $S(k) = x^{2^k}$.

Proof: By simple induction on k .

Basis ($k = 0$): $S(0) = x = x^1 = x^{2^0}$.

Induction: Suppose $S(k) = x^{2^k}$. Then $S(k+1) = S(k)^2 = (x^{2^k}) \bullet (x^{2^k}) = x^{2^k+2^k} = x^{2^{k+1}}$.

We next prove an upper limit on the degree of an ideal viewed as a vertex in the ideal exchange graph. This result is useful in traversing the ideal exchange graph, as is done in [Harper and Bezrukov] to estimate the number of ideals of a given poset P .

Proposition 4.10: Let P be a finite poset and let $\mathcal{J}(P)$ be the distributive lattice of ideals of P . Then the 2-Sperner number of P is an upper bound on the degree of any vertex of (the ideal exchange graph of) $\mathcal{J}(P)$. This upper bound is sharp if P is 2-Sperner and unimodal.

Proof: Let i be an ideal of P . Let L be the antichain of maximal elements of i , and let U be the antichain of minimal elements of the filter $P \setminus i$.

Clearly, $L + U$ is a 2-family in P . By Lemma 4.2, each member of $L + U$ uniquely defines an ideal of P that is adjacent to i in the ideal exchange graph of $\mathcal{J}(P)$. If $x \in L$, then $i \setminus \{x\}$ is an ideal, and if $y \in U$, then $i + \{y\}$ is an ideal. Also by Lemma 4.2, these are all the ideals adjacent to i in the ideal exchange graph of $\mathcal{J}(P)$, and thus the number of ideals adjacent to any ideal i in $\mathcal{J}(P)$ is at most the 2-Sperner number of P .

This upper bound is sharp precisely under the condition that there is an ideal i of P whose sets L and U defined above form a maximum-size 2-family under disjoint union. If P is 2-Sperner and unimodal, we may take $i = \downarrow_L$, where $L = r^{-1}(j)$ and j is the lower of two adjacent ranks of maximum size. This completes the proof.

Finally, we prove that there is an anti-automorphism between the sub-poset B_{kL} of ideals of B_k of ranks 0 through 2^{k-1} and the sub-poset B_{kU} of ideals of B_k of ranks 2^{k-1} through 2^k with the reverse ordering. The anti-automorphism is defined next, using the characteristic vector representation of an ideal introduced earlier.

Definition 4.11: Let $\delta : \mathcal{I}(B_k) \rightarrow \mathcal{I}(B_k)$ be defined by $\delta(i) = (i^R)'$ for each ideal i of B_k , where R is string reversal and $'$ is bitwise complementation (1's complement).

Note that R reverses the bit positions of the characteristic vectors, equivalently reversing the ordering relation in the underlying poset B_k ; $'$ complements the characteristic vectors, equivalently reversing the ordering relation in $\mathcal{I}(B_k)$. Thus δ involves a complementation in both B_k and $\mathcal{I}(B_k)$. The result of applying δ to an ideal i is to dualize i with respect to both B_k and $\mathcal{I}(B_k)$.

Proposition 4.12: δ is “well defined”; i.e., if i is an ideal in $\mathcal{I}(B_k)$, so is $\delta(i)$.

Proof: Let $f : B_k \rightarrow \mathcal{N}_{2^k}$ be the canonical extension of B_k to the totally ordered set $\mathcal{N}_{2^k}$ given by mapping the characteristic vector of an element of B_k to its numeric value as a binary number. Suppose $\delta(i)$ is not an ideal in $\mathcal{I}(B_k)$. Then there is some bit position b that is 1 in $\delta(i)$ and some bit position $d < b$ that is 0 in $\delta(i)$, and such that $f^{-1}(d) < f^{-1}(b)$ in B_k . But then bit position $2^k - 1 - d$ in i is 1 and bit position $2^k - 1 - b < 2^k - 1 - d$ in i is

0. Note that bit positions b and $2^k - 1 - b$ correspond to elements of B_k that are complements of each other; similarly for d and $2^k - 1 - d$. But since $f^{-1}(d) < f^{-1}(b)$, it follows by complementation that $f^{-1}(2^k - 1 - b) < f^{-1}(2^k - 1 - d)$ in B_k . Therefore i is not an ideal in B_k .

The following proposition documents some basic properties of δ . Its proof is trivial.

Proposition 4.13: The following properties are true for δ :

- a) $\delta(i) = (i')^R$
- b) $\delta^2 = 1$; i.e., $\delta(\delta(i)) = i$
- c) $r(\delta(i)) = 2^k - r(i)$

Corollary 4.14: $\mathcal{J}(B_k)$ is rank symmetric; i.e., $W_r = W_{2^k-r}$.

Proof: By (b) of Proposition 4.13, δ is a bijection. By (c) of Proposition 4.13, δ maps ideals of rank r to ideals of rank $2^k - r$.

Theorem 4.15: $i < j$ in $\mathcal{J}(B_k)$ iff $\delta(i) > \delta(j)$ in $\mathcal{J}(B_k)$.

Proof: By (b) of Proposition 4.13, it suffices to show the implication in one direction. Let $i < j$. Since $<$ on $\mathcal{J}(B_k)$ is set inclusion, every bit position of a 1 bit in i must have the corresponding bit position set to 1 in j . This property is preserved by reversing the order of the bit positions (R) in i and j ; every bit position of a 1 bit in i^R must have the corresponding bit position set to 1 in j^R . Equivalently, every bit position of a 0 bit in j^R must have the corresponding bit position set to 0 in i^R . But then, after

complementing to obtain $\delta(j)$ and $\delta(i)$, every bit position of a 1 bit in $\delta(j)$ must have the corresponding bit position set to 1 in $\delta(i)$; i.e., $\delta(i) > \delta(j)$.

Corollary 4.16: δ is an “anti-automorphism” on $\mathcal{I}(B_k)$; i.e., δ is an isomorphism from $\mathcal{I}(B_k)$ to $\mathcal{I}(B_k)^{\text{op}}$, the dual poset of $\mathcal{I}(B_k)$ obtained by reversing the ordering relation.

Proof: Follows immediately from Theorem 4.15 and part (c) of Proposition 4.13.

In particular, the sub-poset B_{kL} of ideals of B_k of ranks 0 through 2^{k-1} is automorphic to the sub-poset B_{kU} of ideals of B_k^{op} of ranks 2^{k-1} through 2^k .

We conclude this chapter by summarizing key results in the literature. As mentioned in Chapter 1, the fundamental result in Sperner theory is (appropriately enough) Sperner’s Theorem [Sperner 1928]: B_k is Sperner; i.e., the width of B_k is the Whitney number $W_{\lfloor k/2 \rfloor} = W_{\lceil k/2 \rceil}$, and the only maximum-size antichain(s) occur(s) at the middle rank(s). Dilworth’s Theorem [Dilworth 1950] extends this by proving that the maximum-size antichain in any poset P equals the minimum size (number of chains) of a partition of P into chains.

Several famous results of order theory pertain to matchings. Philip Hall’s Matching Theorem [Hall 1935] states that a bipartite graph with bipartitions $A + B$ has a matching from A to B iff for all $X \subseteq A$, $|X| \leq |\nabla(X)|$, viewing $A + B$ as a ranked poset with $a \leq b$ precisely when $a - b$ is an edge with $a \in A$ and $b \in B$.

Some basic results of applying algebra to Sperner theory are given below. [Stanley 1991] is a comprehensive study of algebraic approaches to combinatorics.

[Proctor 1982D] shows how algebraic approaches can be applied to the solution of two Sperner problems.

[Greene and Kleitman 1976] showed that B_k is a *symmetric chain order*; i.e., a ranked poset that can be partitioned into saturated chains that are symmetric about the middle rank. This was done by recursively defining a bracketing of (the characteristic vector of) a set as follows: whenever the next unpaired digit after an unpaired 0 is a 1, pair the 0 with the 1. For example, in B_{11} , $\{1, 4, 6, 7, 8, 11\}$ is bracketed 1(0(01)(01)1)10(01). At the end of the process, the substring of unbracketed digits will be in 1^*0^* (i.e., all the 1's occur before all the 0's). Sets in B_k that bracket the same pairs form a saturated chain that is symmetric about the middle rank. In addition to B_k , other symmetric chain orders are the divisors of an integer and the subspaces of a finite-dimensional vector space over a finite field.

Symmetric chain orders are necessarily Peck and thus strongly Sperner. [Griggs 1977] showed that symmetric chain orders are closed under product.

[Griggs 1982] defined a *nested chain order* to be a ranked poset that can be partitioned into pairwise *nested* saturated chains; i.e., for any two chains in the partition, the one with the element of least rank also has the element of greatest rank. Nested chain orders were originally defined in [Gansner]. In [Griggs 1982], it was shown that $C(n, k)$, the subsets of $X = \{1, \dots, n\}$ that intersect $Y = \{1, \dots, k\}$ ordered by inclusion, is a nested chain order and is therefore Sperner.

[Graham and Harper 1969] introduced the notion of normal posets and showed that the NMP condition holds for a ranked, positively weighted poset P iff there is a rank-preserving flowmorphism from P to a chain. This was a key paper that showed the power of the categorical application of morphisms to the analysis of poset properties.

Let $L_{m,n}$ be the set of all monotone-decreasing n -tuples of integers in $\{1, \dots, m\}$. [Harper 1982] defined RUSS morphisms and used them to give a simple proof that $L_{m,n}$ is RUSS.

A sequence $\{s_k\}$ is *2-positive* if, for all k , $s_k \geq 0$ and $s_k s_{k+2} \leq s_{k+1}^2$. Using the above, [Harper 1974] proved the 2-positive Product Theorem: The family of normal posets with 2-positive Whitney numbers is closed under product.

[Proctor, Saks and Sturtevant 1980] proved two key results: The product of two compatible RUSS posets is RUSS, and the product of Peck posets is Peck, where ranked posets P and Q with Whitney numbers $\{p_i\}$ and $\{q_j\}$ are *compatible* if $p_i < p_j \Rightarrow q_{d-i} \leq q_{d-j}$, where d is the maximum rank. This merely says that the size order of the ranks of P is the reverse of the size order of the ranks of Q .

To accomplish this, [Proctor, Saks and Sturtevant 1980] define some terms and prove some intermediate results that are important in their own right. If P is a ranked poset, the complex vector space $\mathbf{P}_i$ is the set of linear combinations (over C) of the elements of the i^{th} rank of P . The complex vector space $\mathbf{P}$ is the disjoint union $\bigoplus_i \mathbf{P}_i$. A *raising (lowering) operator* $\rho (\lambda)$ is a linear transformation on $\mathbf{P}$ such that $\rho(x) (\lambda(x))$ is a linear combination of elements whose rank is $r(x) + 1$ ($r(x) - 1$) involving only elements covering (covered by) x .

In other papers, e.g., [Harper 1982], this definition is specialized to the following condition of interest: A raising (lowering) operator $\rho (\lambda)$ is *Lefschetz* if $\rho (\lambda)$ is completely defined by $\rho(x) = \sum_{x < y} y$ ($\lambda(y) = \sum_{x < y} x$).

[Proctor, Saks and Sturtevant 1980] proved the following results:

(i) If P is a ranked poset with raising operator ρ and lowering operator λ such that, for some integer k , the restriction of λ to P_i is injective for all $i > k$ and the restriction of ρ to P_i is injective for all $i < k$, then P is rank-unimodal and Sperner, and W_k is a Whitney number of maximum size.

(ii) A rank-unimodal poset P is k -Sperner iff $P \times \mathcal{N}_k$ is Sperner [Saks 1979].

(iii) Let P be a finite ranked poset with Whitney numbers $\{W_i\}$. The following are equivalent:

(a) P is RUSS.

(b) For any $i \leq j$, there exists $\min \{W_i, W_j\}$ disjoint saturated chains starting at rank i and ending at rank j .

(c) There exist raising and lowering operators ρ and λ (respectively) such that for $i \leq j$, the *Stanley-Griggs injectivity* property holds:

- if $W_i \geq W_j$, the restriction $\lambda^{j-i}|_j$ of λ^{j-i} to rank j is injective,
- if $W_i \leq W_j$, the restriction $\rho^{j-i}|_i$ of ρ^{j-i} to rank i is injective.

The following algebraic characterization of RUSS posets is proved in [Stanley 1980]:

Let P be a finite graded rank-symmetric poset of rank n . Let $P_i = \{x \in P \mid r(x) = i\}$ and let V_i be the complex vector space with basis P_i . The following three conditions are equivalent:

(i) P is RUSS.

(ii) $\forall 0 \leq i \leq \lfloor n/2 \rfloor$, there exist W_i pairwise disjoint saturated chains $x_i < x_{i+1} < \dots < x_{n-i}$, where W_i is the i^{th} Whitney number.

(iii) $\forall 0 \leq i < n$, there exist linear transformations $\varphi_i: V_i \rightarrow V_{i+1}$ such that

(a) If $0 \leq i \leq \lfloor n/2 \rfloor$, then $\varphi_{n-i-1} \bullet \dots \bullet \varphi_{i+1} \varphi_i: V_i \rightarrow V_{n-i}$ is invertible, and

(b) If $x \in P_i$ and $\varphi_i(x) = \sum_{y \in P_{i+1}} c_y y$, then $c_y = 0$ unless $x < y$.

The last condition simply says that x is a linear combination under φ_i of those elements that cover x .

[Proctor 1982J] defined a necessary and sufficient condition for a poset to be Peck. In this paper, the terms *order raising operator* and *order lowering operator* correspond to the terms *raising operator* and *lowering operator* in [Proctor, Saks and Sturtevant 1980], and a *raising operator* $\rho(x)$ (*lowering operator* $\lambda(x)$) is any linear combination of elements of rank $r(x) + 1$ ($r(x) - 1$), not necessarily obeying the covering relation.

Specifically, given a ranked poset P of maximum rank k , let $\mathbf{P}$ be the direct sum of the vector spaces of formal linear combinations of $\mathbf{P}_i$ over all ranks i . Let H be the linear operator defined by $Hx = (2i - k)x$ for all x of rank i . Then P carries a representation of $\mathbf{SL}(2; C)$ if there exists a lowering operator λ and an order raising operator ρ on $\mathbf{P}$ such that $\rho\lambda - \lambda\rho = H$. ($\mathbf{SL}(2; C)$ is the special linear group of invertible 2×2 matrices with complex entries having determinant 1.) The following result was proved in [Proctor 1982J]:

Let P be a ranked poset, and let ρ be an order raising operator on P . Then there is at most one lowering operator λ on P corresponding to ρ such that P carries a representation of $\mathbf{SL}(2; C)$.

This result was used to prove that the product of Peck posets is Peck. Since [Stanley 1980] proved that all *Bruhat posets* (orders defined on *Coxeter groups*) are Peck, products of Bruhat posets are also Peck.

[Harper 1982] cast the algebraic approach of [Proctor, Saks and Sturtevant 1980] in categorical terms. If V and W are vector spaces and $\rho : V \rightarrow V'$ and $\rho' : W \rightarrow W'$ are linear transformations, a linear transformation $\phi : V \rightarrow W$ is an *intertwining operator* iff ϕ commutes with respect to ρ and ρ' ; i.e., $\phi\rho = \rho'\phi$. [Harper 1982] proved that intertwining operators preserve kernels and then proved the following key result:

Intertwining operators that satisfy a specific algebraic condition, discussed in Chapter 10 of this paper, are closed under composition and are the morphisms of a category of linear operators.

[Harper 1982] used this to define the category **RUSS** of objects (P, ρ, λ) , where P is a graded poset, ρ is a raising operator, and λ is a lowering operator. A *RUSS-morphism* f is a rank-preserving order morphism $f : (P, \rho, \lambda) \rightarrow (P', \rho', \lambda')$ such that

- f and ρ give an intertwining operator $f_r : \rho \rightarrow \rho'$
- f and λ give an intertwining operator $f_l : \lambda \rightarrow \lambda'$

[Harper 1982] then proved that RUSS-morphisms are indeed the morphisms of the category **RUSS**; i.e., if P is **RUSS** with raising operator ρ and lowering operator λ , and f is a RUSS-morphism $f: (P, \rho, \lambda) \rightarrow (P', \rho', \lambda')$, then P' is also **RUSS**.

Since the category of graded posets is a functor category over **Set_{fin}**, [Harper 1982] showed that a restriction of the category **RUSS** has pushouts and coequalizers. It follows as a corollary that a (finite) Boolean algebra modulo a permutation group on the generators (i.e., the atoms) is RUSS.

CHAPTER 5: FLOW ALGORITHMS, NORMALITY AND SPERNERITY

Flows on networks have proven very useful in analyzing combinatorial problems. This chapter describes flow algorithms on networks in general, and one flow algorithm in particular that we used to establish Sperner properties of specific posets in this study.

Given a network, the *max flow problem* is the problem of assigning flows to the edges (preserving all flow properties such as conservation of flow) to achieve the maximum flow through the network (equivalently, the max flow out of the start vertex s , or the max flow into the terminal vertex t). Applications of the max flow problem abound. The *min flow* problem is an obvious variant of the max flow problem, where each edge must carry at least as much flow as the edge weight.

A famous theorem proves that the max flow is equal to the *min cut*; i.e., a cut the sum of whose edge flows is minimum over all cuts. [Ford and Fulkerson 1962] is the classic reference. The dual for min flow problems is that min flow = max cut.

Another variant of this problem is of interest to the study of Sperner properties of posets. The capacities, or *weights*, of a network may be on the vertices instead of the edges. The following standard transformation converts this problem into the edge-weighted problem. Split each vertex v into an in-vertex v_i and an out vertex v_o . Add an edge from v_i to v_o with capacity equal to the weight of v . All edges that originally came into v now come into v_i , and all edges that originally came out from v now come out of v_o . These edges all have infinite capacity (or 0 capacity in the min flow case) so that the flow constraints are imposed solely by the edges joining the input and output copies of the vertices.

A further simplification is possible in case the poset of interest is bipartite. In this case, the vertices do not need to be split. It is sufficient to add vertices s and t with edges and capacities set up as follows. Each vertex v in the “lower” bipartition (such that $v < w$ for some w in the “upper” bipartition) has an edge incoming from s whose capacity is equal to the weight of v . Similarly, each vertex w in the upper bipartition has an edge out to t whose capacity is equal to the weight of w . The edges of the original bipartite poset are unconstrained (with infinite or 0 capacity as appropriate). Thus a vertex in the lower bipartition is saturated in the poset iff the flow into it from s equals the edge capacity; similarly for a vertex in the upper bipartition and its flow into t .

The test for (strong) Spernerity of a ranked poset P reduces to a min flow problem on the Hasse diagram of P . Recall that a cut is the edge set between a cutset and its complement. For a nontrivial ideal of a ranked poset (which is clearly a cutset), all the edges in a cut of the Hasse diagram go from a maximal vertex in the cutset to a minimal vertex in the complement. L. H. Harper [Harper 1974] has shown that, for a weighted poset, the maximal vertices of a cutset with a max cut is an antichain of largest weight. Thus the Sperner problem on a weighted poset is equivalent to the min flow problem on its Hasse diagram. An immediate corollary is that a ranked poset is Sperner if there is a max cut whose maximal elements all have the same rank. Since a ranked poset P is strongly Sperner iff $P \times \mathcal{N}_k$ is Sperner for $k = 1, \dots, (\max \text{rank}(P) + 1)$, a solution of the dual min-flow problem on $P \times \mathcal{N}_k$ for all k provides a test for strong Spernerity of P .

Another test for strong Spernerity involves testing for normalized flows. L. H. Harper [Harper 1980] has shown that a normal poset is strongly Sperner. Thus a stronger result may be obtained by using a flow algorithm to show that a specific poset is normal.

The test for normality uses the property that a ranked, positively weighted poset $(P, \leq, w)$ is normal iff there is a rank-preserving flowmorphism from P to a chain [Graham and Harper 1969]. This reduces to the problem of finding a normalized flow between each pair of adjacent ranks.

Thus the test for normality is faster and easier than that for Spernerity even though the condition for normality is more challenging to meet. This is because the condition for normality applies to bipartite flows between each pair of adjacent ranks, which can be tested in parallel, whereas the condition for strong Spernerity applies to $(\max \text{rank } P + 1)$ global flows that cannot be decomposed into smaller subflows.

A further simplification is possible. L. H. Harper [Harper 1974] showed that $\mathcal{J}(B_k)$ is normal iff the quotient poset $\mathcal{J}(B_k) / S_k$ is normal, where S_k is the symmetric group on k generators and the quotient poset is the result of the action of S_k on B_k . This follows from the existence of pushouts in the category **FLOW**. The weight of each vertex in $\mathcal{J}(B_k) / S_k$ is the cardinality of the equivalence class.

Informally, S_k acts on B_k by defining two ideals i and j to be in the same equivalence class iff i and j are isomorphic, in the sense made precise in the next few paragraphs. This is true if and only if j is a permutation of i . An edge exists from equivalence class $[a]$ to equivalence class $[b]$ iff there exist x in $[a]$ and y in $[b]$ such that $x < \bullet y$; i.e., $y = x + \{c\}$ for some minimal vertex c in $B_k \setminus \{x\}$.

More precisely, a permutation p of B_k is defined as the action of a permutation on the k atoms. Since every element b of B_k is the join of all atoms $a \leq b$, p is naturally extended to all of B_k . Viewing the canonical representation of b as the join of the characteristic vector of all atoms below b shows this. In B_3 , the atoms are 001, 010, and 100, and an element (e.g., 011) can be unambiguously expressed as the join of atoms 001 and 010. Let p be the permutation $001 \rightarrow 010 \rightarrow 100 \rightarrow 001$; i.e., $p(001) = 010$, $p(010) = 100$, and $p(100) = 001$. Then p acts on B_3 to give $p(011) \rightarrow 110$ by permuting each of the constituent atoms to its new position. Viewing the bit representations of the elements as numbers, we see that the effect of p is additive; i.e., $p(3) = p(1) + p(2) = 2 + 4 = 6$. The canonical representation of the elements of B_k as the numeric value of these bit representations guarantees this will always work.

As with any function, $p(s) = \{p(x) \mid x \in s\}$ for any subset s of B_k . In this way, $p(i)$ is well-defined for any given ideal i as the permutation p applied to all members of i . Using this, we can define an equivalence $\sim$ on $\mathcal{I}(B_k)$ by $i \sim j$ iff $p(j) = i$ for some permutation p . This is the action of the symmetric group S_k on $\mathcal{I}(B_k)$. Writing the ideals as characteristic vectors in (reverse) lex order allows us to identify the positions of the atoms at $2^0, 2^1, \dots$. Then the action of a permutation p on an ideal i is completely defined by its action on the positions of the atoms; the other positions are permuted as composites of these positions just as for B_k . Then $i \sim j$ can be tested by finding a permutation p of (the characteristic vector of) j so that $p(j) = i$.

The advantage of using the quotient poset is its significantly reduced memory requirements. This is achieved at the expense of the added cost to perform the

isomorphism tests. Note that there is no known easy way to generate the quotient, as this is an instance of the graph isomorphism problem, which may be $\mathcal{NP}$ -complete.

Using all of the above, a flow-based algorithm for testing $\mathcal{J}(B_k)$ for strong Spernerity proceeds as follows:

- Generate $\mathcal{J}(B_k)$. Algorithms for doing this are given in Chapters 6 and 7.
- Build the weighted quotient poset. In our case, this is $\mathcal{J}(B_k) / S_k$ as defined above.
- Use a bipartite flow algorithm to find a normalized (unit) flow, if one exists, between each pair of adjacent ranks as described next.

The requirement that a unit flow exist between each pair of successive ranks requires fractional capacities and flows on edges, which is computationally cumbersome, hard to check, and prone to rounding error. Specifically, suppose the Whitney number of rank r is W_r . Then a vertex v of rank r and weight $W_{r,v}$ must be normalized so that its weight is $W_{r,v} / W_r$, and thus the sum of the weights over all vertices of rank r is 1.

To counteract this issue, the normal flow problem for ranks r and $r + 1$ is converted into an integer bipartite flow problem by scaling the weights up by $W_r \times W_{r+1}$. Then each vertex v of rank r has an incoming edge from the start vertex s with capacity $(W_{r,v} / W_r) \bullet (W_r \times W_{r+1}) = W_{r,v} \times W_{r+1}$, and each vertex w of rank $r + 1$, has an outgoing edge to the terminal vertex t with capacity $(W_{r+1,w} / W_{r+1}) \bullet (W_r \times W_{r+1}) = W_{r+1,w} \times W_r$. The edges between vertices in the bipartite network have infinite capacity. There is then a normal flow between ranks r and $r + 1$ iff there is a flow of $W_r \times W_{r+1}$ on this scaled version of the network, with all integer capacities and flows. Equivalently, the flow out

of s , and the flow into t , must be $W_r \times W_{r+1}$. This will be the case iff each edge out of s and each edge into t is saturated.

Since a normalized flow must equal the product of the Whitney numbers of the adjacent ranks under test, either the max flow or the min flow variant may be used. The max flow variant is chosen because the initial condition (all edge flows = 0) is immediate.

There are many algorithms for finding the max (min) flow on a network. One that is comparatively efficient is the *MPM algorithm* [Malhotra, Pramodh Kumar and Maheshwari 1978]. The definition of the MPM algorithm in the remainder of this Chapter is paraphrased from [Even 1979], an excellent reference.

The max flow problem will be solved independently for each pair of adjacent ranks of the quotient poset. In the following, the network is a fixed pair of adjacent ranks. It is important to note that, since these bipartite flow problems are independent, they can be solved in parallel by running multiple copies of the program.

Let e be an edge $u \rightarrow v$ in the network. Edge e is *useful in the forward direction* (or *useful from u to v*) if $f(e) < c(e)$, where $f(e)$ is the flow on e and $c(e)$ is the capacity of e . Edge e is *useful in the backward direction* (or *useful from v to u*) if $f(e) > 0$. If e is useful in the forward direction, flow can be added to e . If e is useful in the backward direction, flow can be taken away from e .

As is typical for flow algorithms, the MPM algorithm augments flow from an initial condition (no flow) until no further augmentation is possible. The MPM algorithm accomplishes this in *phases*. Each phase starts with an existing flow, builds a *layered*

network, calculates an *augmenting flow* on the layered network, and ends by adding the calculated augmenting flow to the existing flow.

The layered network for a phase is built as follows:

- 1 Set $V_0 = \{s\}$ and $i = 0$.
- 2 Set $T = \{v \mid v \notin V_j \text{ for } j \leq i \text{ and there is a useful edge from a vertex in } V_i \text{ to } v\}$.
- 3 If $T = \emptyset$, halt; the current total flow is maximum.
- 4 If $t \in T$, set $l = i + 1$, set $V_l = \{t\}$ and halt.
- 5 Set $V_{i+1} = T$, increment i and go to step 2.

The sets V_i are the *layers* of the layered network. For $1 \leq i \leq l$, let E_i be the set of edges from a vertex in V_{i-1} to a vertex in V_i . Note that an edge e may be useful in either the forward direction or the backward direction. If, in the layered network, $e = u \rightarrow v$, where $u \in V_{i-1}$ and $v \in V_i$ for some i , e is used in the forward direction; otherwise e is used in the backward direction. All edges of E_i in the layered network are considered to be directed from V_{i-1} to V_i .

Note that each pass from step 2 through step 5 either a) halts in step 3 because the layered network cannot be completed, b) halts in step 4 when t is added to a layer, or c) reduces the set of vertices not yet assigned to a layer by at least 1. Thus the algorithm halts. The algorithm investigates each edge at most twice (once in each direction). Thus the layered network algorithm is $O(|E|)$, where E is the edge set of the bipartite network.

If the above algorithm halts in step 3, the flow is maximum. (See [Even 1979] for a proof.) If the above algorithm halts in step 4, the layered network has been built for this phase.

The existing flow and capacity for edge e are respectively denoted $f(e)$ and $c(e)$. The next step is to compute an augmenting flow. Define $c'(e) = c(e) - f(e)$ if e is used in the forward direction in the layered network, $c'(e) = f(e)$ if e is used in the backward direction in the layered network. Then set $f'(e) = 0$ for each edge e and calculate an augmenting flow on the layered network.

The calculation of the augmenting flow will be discussed later. For now, its key property will be described. An edge e in the layered network is *saturated* if $f'(e) = c'(e)$. A flow is *maximal* iff every path in the layered network from s to t has at least one saturated edge. The augmenting flow that is calculated will be a maximal flow, but not necessarily the maximum flow on the network. The augmenting flow is used to update the existing flow by updating the flow of each edge e :

If e is used in the forward direction, set $f(e) \leftarrow f(e) + f'(e)$.

If e is used in the backward direction, set $f(e) \leftarrow f(e) - f'(e)$.

The entire max flow algorithm ends when a new layered network cannot be built. The number of phases necessary to reach this condition is bounded above by $|V|$, the number of vertices in the bipartite network. See [Even 1979] for a proof.

The MPM flow algorithm will now be described. Let N' be the layered network with vertices V' . For each $v \in V'$, let $\alpha(v) = \{e_{u_1}^I, \dots, e_{u_{p_v}}^I\}$ be the edges coming into v in the layered network from $u_1, \dots, u_{p_v}$, respectively, and let $\beta(v) = \{e_{w_1}^O, \dots, e_{w_{q_v}}^O\}$ be the edges going out of v in the layered network to $w_1, \dots, w_{q_v}$, respectively.

For each vertex v , the algorithm computes the *in-potential* $IP(v)$ and *out-potential* $OP(v)$. $IP(v)$ is a local upper bound on the flow that can come into v , and $OP(v)$ is a local upper bound on the flow that can go out from v . If $v \neq s$ and $v \neq t$, the *potential* $P(v) = \min (IP(v), OP(v))$; $P(s) = OP(s)$ and $P(t) = IP(t)$.

The algorithm then finds a vertex v with minimum potential $P(v)$. This potential is pushed out to t and pulled back to s , using all the potential in v . To do this, we use the edges of $\beta(v)$ one by one, saturating each edge as long as potential flow from v remains. This pushes flow to the next higher layer. The OF value for each vertex that receives flow from v keeps track of the flow that must be pushed toward t . This process is repeated for each vertex that receives flow until all of $P(v)$ has been pushed to t . This is done in steps 6 through 8.

An analogous process starts with the edges of $\alpha(v)$ and pulls all of $P(v)$ back to s . This is done in steps 9 through 11. The IF value for each vertex that gives flow toward v keeps track of the flow that must be pulled toward s .

The choice of v guarantees that each vertex that pushes or pulls flow has sufficient capacity to do so. For each vertex w , $IF(w)$ and $OF(w)$ keep track of the flow that must be pulled into or pushed out of w , respectively. The capacities $IP(w)$ and $OP(w)$ are updated as flows are assigned to edges into or out of w , respectively.

Once the potential flow for v has been pushed to t and pulled to s , potentials are re-computed, a new minimum-potential vertex v is found and flow is again pushed to t and pulled to s . This process terminates when either s or t has 0 potential.

The algorithm below is taken directly from [Even 1979]. The step numbers are listed as comments in the normality test program. See Chapter 8 for a discussion of the normality test program and Appendix B for a listing of the normality test program.

- 1 For every $v \in V'$, $IP(v) \leftarrow \sum_{i=1}^{pv} c'(e_{vi}^I)$, $OP(v) \leftarrow \sum_{i=1}^{qv} c'(e_{vi}^O)$, $m_v \leftarrow 1$, $m_v' \leftarrow 1$.
- 2 For every e in N' , $f'(e) \leftarrow 0$.
- 3 Perform the following operations:
 - 3.1 $P(s) \leftarrow OP(s)$, $P(t) \leftarrow IP(t)$.
 - 3.2 For $v \in V' \setminus \{s, t\}$, $P(v) \leftarrow \min(IP(v), OP(v))$.
 - 3.3 Find a vertex v for which $P(v)$ is minimum.
- 4 If $P(v) = 0$, perform the following operations:
 - 4.1 If $v = s$ or $v = t$, halt; the present f is maximal.
 - 4.2 For every m from m_v to p_v , where $e_{vm}^I = u - v$, if $u \in V'$ then

$$OP(u) \leftarrow OP(u) - (c'(e_{vm}^I) - f'(e_{vm}^I)).$$
 - 4.3 For every m' from m_v' to q_v , where $e_{vm'}^O = v - u$, if $u \in V'$ then

$$IP(u) \leftarrow IP(u) - (c'(e_{vm'}^O) - f'(e_{vm'}^O)).$$
 - 4.4 $V' \leftarrow V' \setminus \{v\}$ and go to step 3.
- 5 Find i for which $v \in V_i$. Let $j \leftarrow i$, $k \leftarrow i$. Also $OF(v) \leftarrow P(v)$, $IF(v) \leftarrow P(v)$ and for every $u \in V' \setminus \{v\}$ let $OF(u) \leftarrow 0$ and $IF(u) \leftarrow 0$.
- 6 If $k = l$ (the pushing from v to t is complete) go to step 9.
- 7 Assume $V_k = \{v_1, \dots, v_k\}$.
 - 7.1 $r \leftarrow 1$.
 - 7.2 $u \leftarrow v_r$.
 - 7.3 If $OF(u) = 0$ then go to step 7.5.
 - 7.4 ($OF(u) > 0$. Starting with $e_{um'}^O$, we push the excess supply towards t .)
 - 7.4.1 $e \leftarrow e_{um'}^O$ and assume $e = u - w$.
 - 7.4.2 If $f'(e) = c'(e)$ or $w \notin V'$ then go to step 7.4.5.
 - 7.4.3 If $OF(u) \leq c'(e) - f'(e)$ then perform the following operations:

$$\begin{aligned} f'(e) &\leftarrow f'(e) + OF(u) \\ OF(w) &\leftarrow OF(w) + OF(u) \\ OP(u) &\leftarrow OP(u) - OF(u) \\ IP(w) &\leftarrow IP(w) - OF(u) \end{aligned}$$
 and go to step 7.5.
 - 7.4.4 ($OF(u) > c'(e) - f'(e)$)

$$\begin{aligned} OF(u) &\leftarrow OF(u) - (c'(e) - f'(e)) \\ OF(w) &\leftarrow OF(w) + (c'(e) - f'(e)) \\ OP(u) &\leftarrow OP(u) - (c'(e) - f'(e)) \\ IP(w) &\leftarrow IP(w) - (c'(e) - f'(e)) \\ f'(e) &\leftarrow c'(e) \end{aligned}$$

7.4.5 $m'_u \leftarrow m'_u + 1$ and go to step 7.4.1.
 7.5 If $r = k'$, go to step 8.
 7.6 $r \leftarrow r + 1$ and go to step 7.2.
 8 $k \leftarrow k + 1$ and go to step 6.
 9 If $k = 1$ (the pulling from s to v is complete too) then $V' \leftarrow V' \setminus \{v\}$ and go to step 3.
 10 Assume $V_j = \{v_1, \dots, v_j\}$.
 10.1 $r \leftarrow 1$.
 10.2 $u \leftarrow v_r$.
 10.3 If $IF(u) = 0$ then go to step 10.5.
 10.4 ($IF(u) > 0$. Starting with $e_{um_u}^I$, we pull the excess demand towards s .)
 10.4.1 $e \leftarrow e_{um_u}^I$ and assume $e = w - u$.
 10.4.2 If $f'(e) = c'(e)$ or $w \notin V'$ then go to step 10.4.5.
 10.4.3 If $IF(u) \leq c'(e) - f'(e)$ then perform the following operations:
 $f'(e) \leftarrow f'(e) + IF(u)$
 $IF(w) \leftarrow IF(w) + IF(u)$
 $IP(u) \leftarrow IP(u) - IF(u)$
 $OP(w) \leftarrow OP(w) - IF(u)$
 and go to step 10.5.
 10.4.4 ($IF(u) > c'(e) - f'(e)$)
 $IF(u) \leftarrow IF(u) - (c'(e) - f'(e))$
 $IF(w) \leftarrow IF(w) + (c'(e) - f'(e))$
 $IP(u) \leftarrow IP(u) - (c'(e) - f'(e))$
 $OP(w) \leftarrow OP(w) - (c'(e) - f'(e))$
 $f'(e) \leftarrow c'(e)$
 10.4.5 $m_u \leftarrow m_u + 1$ and go to step 10.4.1.
 10.5 If $r = j'$, go to step 11.
 10.6 $r \leftarrow r + 1$ and go to step 10.2.
 11 $j \leftarrow j - 1$ and go to step 9.

Inspection of the above algorithm shows that, if all capacities are integers, then all computations are additions or subtractions of integer quantities. Therefore, the resulting flows on the edges, and the total flow out of s , are integers.

To analyze the run-time complexity of the MPM algorithm, observe that each edge can be saturated at most once, either in step 7.4.4 or in step 10.4.4. The other uses of edges are in steps 7.4.3 and 10.4.3 and can be bounded as follows:

Fix v and consider the case in which $P(v)$ is minimum. This can happen at most once for v . For every $u \neq v$, at most one outgoing edge is used in 7.4.3 without saturating it, and at most one incoming edge is used in 10.4.3 without saturating it. Thus, the number of uses of edges is bounded above by $|E| + |V|^2 = O(|V|^2)$. This is the complexity of finding a maximal flow in the layered network for a given phase.

As noted earlier, the number of phases is bounded above by $|V|$ and the complexity of building the layered network for a given phase is $O(|E|)$. Therefore, the MPM max flow algorithm is $O(|V| \bullet |E|) + O(|V|^3) = O(|V|^3)$.

CHAPTER 6: THE TRADITIONAL IDEAL GENERATION ALGORITHM

As discussed in Chapter 5, the general approach to algorithms for determining Sperner properties of ideal posets is to generate the ideals (represented by their characteristic vectors), then run flow algorithms to test for strong Spernerity or normality. A widely-used ideal generation algorithm will be given with a correctness proof.

As noted in Chapter 5, the test for normality involves finding normalized flows between each pair of adjacent ranks. A key point is that these normalized flows are independent of each other. Thus it is possible to construct copies of the normal test program and run them in parallel. This represents a very significant calendar time savings in comparison with the time required to test for strong Spernerity, which requires multiple min flow calculations on entire large networks. It thus becomes valuable to find a way to efficiently generate only those ideals that have specified ranks. These ideals will henceforth be called “ideals of interest”. A selective ideal generation algorithm that generates only ideals of interest is given (again with a correctness proof) in this chapter. This algorithm is based on the traditional ideal generation algorithm.

Chapter 7 presents a new algorithm, the ideal exchange algorithm, with a proof of correctness. The ideal exchange algorithm provides a slight improvement over the selective ideal generation algorithm in calendar computation time when running the normal test algorithm in parallel. The results for $\mathcal{J}(B_1)$ through $\mathcal{J}(B_6)$ presented in Chapter 9 compare the execution times of the normal test algorithm under various conditions and show the speed advantage of running the normal test algorithm in parallel.

The traditional ideal generation algorithm, presented first, generates the ideals $\mathcal{I}(P)$ of a given poset P . This algorithm was published in [Schrage and Baker 1978].

The input is $(P, \leq)$, given in any fashion that allows for determination of membership in P and test for $\leq$. The output is an array $I = \mathcal{I}(P)$ and the number of ideals $n_I = |I|$. Each member of I is the characteristic vector of an ideal of P .

The algorithm has the additional property that the result I is already sorted in numerical order (treating each characteristic vector as a binary number). Equivalently, the characteristic vectors, viewed as strings over $\{0, 1\}$, are ordered *lexicographically* (*lex order*) in $\mathcal{I}(P)$. This is useful in its own right, and is important in proving the correctness of the ideal generation algorithm, which follows:

ALGORITHM: Ideal Generation Algorithm

INPUT: Finite poset $(P, \leq)$

OUTPUT: Array I of characteristic vectors of ideals of P in lex order; $n_I = |I(P)|$

- 1 Define a total extension $f: P \rightarrow \mathcal{N}_{|P|}$ so that, if $u \leq v$ in P , then $f(u) \leq f(v)$ in $\mathcal{N}_{|P|}$.
- 2 For each i in $\mathcal{N}_{|P|}$, construct a list $L[i]$ of successors j such that $f^{-1}(i) <_{\bullet} f^{-1}(j)$ in P .
- 3 Initialize $I[0] = 0$; at the end of the algorithm, I will have all ideals of P in lex order.
- 4 Initialize $n_I = 0$; n_I is the index of the ideal most recently added to I .
- 5 Repeat
 - 5.1 Find the lowest order 0 bit b in $i = I[n_I]$, the most recently generated ideal.
 - 5.2 Set j to the result of setting bit b of i to 1, leaving all other bits the same.
 - 5.3 For each bit d in j to the right of b , moving left to right from $b - 1$ to 0,
 - 5.3.1 If no member of $L[d]$ has its bit set to 1 in j , set bit d to 0 in j .
 - 5.4 Increment n_I .
 - 5.5 Set $I[n_I] = j$.
 - 5.6 Until $I[n_I] = 2^{|P|} - 1$.
- 6 Increment n_I ; n_I now equals the number of ideals in I .

We start with five observations, then break the proof of correctness of the ideal generation algorithm into several lemmas that will be useful in proving the correctness of the selective ideal generation algorithm.

The first observation is that the extension f merely numbers the elements of P so that, by listing elements of P in the order given by f , smaller elements of P are listed before larger elements of P . If $P = B_k$ for some k , the identity map provides a canonical extension that is used in applying this algorithm to $\mathcal{I}(B_k)$. For example, if $k = 3$, the canonical extension f is $000 < 001 < 010 < 011 < 100 < 101 < 110 < 111$.

The second observation is that the bit positions b and d in steps 5.2, 5.3, and 5.3.1 indicate whether the element $f^{-1}(b)$ or $f^{-1}(d)$ of P is in the most recently generated ideal i ($b = 1$ or $d = 1$) or not ($b = 0$ or $d = 0$).

The third observation is that the ideal generation algorithm never depends on any ideal generated earlier than the most recently generated ideal. One implication of this is that, depending on the application, ideals can be processed as they are generated and need not be stored in the array I . This saves considerable storage (perhaps millions of bytes).

The fourth observation is that the ideal generation algorithm works for any poset P ; in particular, P does not have to be ranked. However, the ideals generated by the algorithm form a ranked poset (a distributive lattice, in fact, by Proposition 4.6) ordered by set inclusion. By Proposition 4.4, the rank $r(i)$ of an ideal i is its cardinality.

The fifth observation is that the left-to-right order in steps 5.3 and 5.3.1 is important. If the search were right-to-left, the ideal test would sometimes fail when it should pass. For example, consider the ideal 00001111 in B_3 . The next ideal should be 00010001, and is correctly determined by the ideal generation algorithm by setting bit 4 to 1, then setting bits 3, 2, and 1 to 0 in turn (because bit 4 corresponds to an atom). However, if the search were made right-to-left, the next ideal generated would be

00010111, because bits 1 and 2 would each be blocked by bit 3 (since $1 = 001 < 011 = 3$, $2 = 010 < 011 = 3$, and bit 3 has not yet been set to 0).

The proof of correctness of the ideal generation algorithm follows.

Lemma 6.1: If i is an ideal and b is the lowest-order 0-bit of i , then the result j obtained by setting bit b to 1 and leaving all other bits unchanged in i is an ideal.

Furthermore, j is the smallest ideal in lex order with the following two properties:

- (a) $j > i$
- (b) $r(j) > r(i)$

Proof: Suppose j is not an ideal. Then there exist x in j and y not in j with $y < x$. Equivalently, there is a 1 in bit position $f(x)$ of j , a 0 in bit position $f(y)$ of j , bit position $f(y)$ is smaller (lower-order) than bit position $f(x)$, and $y < x$. If $b \neq f(y)$ and $b \neq f(x)$, these conditions also hold in i , contradicting the assumption that i is an ideal. Since bit b is 1 in j , $b \neq f(y)$. The only other possibility is that $b = f(x)$, and bit $f(y) < b$ is 0. But this contradicts the assumption that b is the lowest-order 0-bit in i .

Therefore j is an ideal. Clearly, $j > i$ and $r(j) > r(i)$; in fact, $r(j) = r(i) + 1$.

Suppose j is not the smallest ideal satisfying (a) and (b). Then there exists p such that $i < p < j$ and $r(p) > r(i)$. Note that i and j agree on all bits higher-order than b . If p differs from i or j on any of these bits, then either $p < i$ or $p > j$, a contradiction. Therefore, p differs from j only in the $b + 1$ lowest-order bits 0 through b . But these bits are all 1 in j , so $r(p) < r(j)$ and thus $r(p) \leq r(i) = r(j) - 1$, a contradiction. This completes the proof.

Corollary 6.2: The characteristic vector i with the r lowest bits all 1 and the remaining bits all 0 is the smallest ideal in lex order of rank r .

Proof: By Lemma 6.1, i is an ideal. It is clearly the smallest ideal of rank r in lex order.

Lemma 6.3: If i is an ideal, the result j of changing any one bit position b from 1 to 0 and leaving all other bits unchanged is an ideal iff, for all d satisfying $f^{-1}(b) < f^{-1}(d)$, bit position d is 0.

Proof: If j is an ideal with bit $b = 0$, it must be the case that bit position d is 0 for every d satisfying $f^{-1}(b) < f^{-1}(d)$.

Conversely, suppose j is not an ideal. As in the proof of Lemma 6.1, for some x and y , there is a 1 in bit position $f(x)$ of j , a 0 in bit position $f(y)$ of j , bit position $f(y)$ is smaller (lower-order) than bit position $f(x)$, and $y < x$. If $b \neq f(y)$ and $b \neq f(x)$, these conditions also hold in i , contradicting the assumption that i is an ideal. Since bit b is 0 in j , $b \neq f(x)$. Then it must be that $b = f(y)$. Let $d = f(x)$. Then $f^{-1}(b) = y < x = f^{-1}(d)$, and bit position $d = f(x)$ is 1.

Lemma 6.4: The ideal generation algorithm generates only ideals, and they are generated in lex order.

Proof: The ideal generation algorithm starts by generating 0, the smallest ideal of any poset, in step 3. Lemmas 6.1 and 6.3 show that the result of each transformation in the ideal generation algorithm is an ideal. It remains to show that the ideals generated by the ideal generation algorithm are generated in lex order. The proof is by induction on n_i , the index of the most recent ideal generated. Step 3 provides the basis $I[0] = 0$.

Suppose the first p ideals generated by the algorithm are in lex order, and let i be the p^{th} ideal generated. The next ideal j will be generated in steps 5 through 5.6. These

steps manipulate only the $b + 1$ lowest-order bits 0 through b . From steps 5.1 and 5.2, bit b is 0 in i and 1 in j . Therefore $i < j$, and the induction is extended.

Lemma 6.5: Given a finite poset P , the ideal generation algorithm terminates in a finite number of steps.

Proof: Step 1 is an effective procedure defined from a finite domain to a finite codomain. Step 2 builds a finite list for each member of a finite set. Steps 3, 4 and 6 are each $O(1)$.

This leaves step 5 and its sub-steps. There are four nested loops in this portion of the algorithm, as follows:

- generate the next ideal
 - search bits right-to-left in the most recently generated ideal to find the lowest-order 0
 - search bits left-to-right from this bit position
 - for each bit position, search its higher-order successor bits for any 1 bits

The three inner loops are all bounded by $|P|$, the length of the characteristic vector. At the completion of the three inner loops, the algorithm either generates a new ideal or terminates. Since the number of ideals for a finite poset P is bounded above by $2^{|P|}$, the algorithm must eventually terminate.

Lemma 6.6: If i is an ideal, then i is generated by the ideal generation algorithm.

Proof: The ideal generation algorithm starts by generating the smallest ideal, 0, in step 3. By Lemma 6.5, the ideal generation algorithm terminates; from step 5.6, the last ideal it generates is the largest ideal, $2^{|P|} - 1$. By Lemma 5.4, the ideals generated by the

ideal generation algorithm are generated in lex order. It remains to show that the ideal generation algorithm generates all ideals in between without skipping over any ideals.

Suppose there is at least one ideal that is not generated by the ideal generation algorithm. Let p be the smallest such ideal in lex order, and let i be the ideal generated by the ideal generation algorithm immediately preceding p in lex order. Steps 5.1 and 5.2 change the lowest-order 0-bit b of i to 1 and leave the remaining bits unchanged. By Lemma 6.1, the result (call it a) is an ideal. By steps 5.3 through 5.5, the next ideal j generated after i by the ideal generation algorithm satisfies $i < p < j \leq a$. Thus i, p, j and a all agree on all bit positions greater than b . Since bit positions 0 through $b - 1$ of i are all 1, bit position b must be 1 for both p and j . Since $p < j$, the highest-order bit position d where p and j differ must be 0 in p and 1 in j ; furthermore, $d < b$. Since step 5.3 visits the bit positions left to right (i.e., from higher-order to lower-order) from b , bit position d is the first bit visited in this step where p and j differ. Since p is an ideal, all bits t in $L[d]$ must be 0 in p , and thus in j , because each such t has a higher-order bit position than d . But then step 5.3.1 will set d to 0, contradicting the assumption that j is the next ideal generated by the ideal generation algorithm. This completes the proof.

Theorem 6.7: The ideal generation algorithm is correct. That is,

- a) Upon completion of the ideal generation algorithm, $I = \mathcal{J}(P)$ and $n_I = |I|$, and
- b) The ideal generation algorithm terminates in a finite number of steps.

Moreover,

- c) I is in lex order; i.e., $i < j \Rightarrow I[i] < I[j]$.

Proof:

- a) Lemmas 6.4 and 6.6.
- b) Lemma 6.5.
- c) Lemma 6.4.

In analyzing flows between adjacent ranks of $\mathcal{I}(B_k)$, the question naturally arises whether the ideal generation algorithm can be modified to generate only the ideals of interest. One way to accomplish this would be to add a test just before step 5.5 of the traditional ideal generation algorithm to see if the ideal is of interest. This would save considerable space.

The algorithm can also be modified to save significant time as well. This can be done by bounding the search on the ranks of ideals processed by the algorithm, at the low end of the ideals of interest, and by terminating the algorithm when the maximum ideal of interest is generated, at the high end. We consider these two points in turn, then define a selective ideal generation algorithm that defines only ideals of interest and is more efficient in run time than the traditional ideal generation algorithm.

To bound the ideals processed at the low end, consider how the traditional ideal generation algorithm processes ideals. The ranks of ideals do not play a part in the algorithm, and the ranks of ideals do not follow an obvious pattern in the order that the ideals are generated by the traditional ideal generation algorithm.

Let i be an ideal of rank m , the minimum rank of interest. Recall from Lemma 6.1 that the next ideal j in lex order with $r(j) > m$ is formed by setting the least significant 0 bit of i to 1. Then $r(j) = m + 1$. There may be several ideals between i and j in lex order, but none of them has rank greater than m . The trick is to find all ideals between i and j

whose rank is exactly m and generate them in lex order without processing any ideals of lower rank.

Suppose the ideals of rank m between i and j are as follows in lex order:

- $i_0 = i$ bit $b_0 = b$ is 0
- i_1 bit $b_1 < b_0$ is 0
- .
- .
- .
- i_q bit $b_q < b_{q-1}$ is 0

There may be several other ideals of smaller rank between i and j . It is also worth noting that j immediately follows i_q in lex order. Now the ideal generation algorithm converts i to j by setting the least significant 0 bit b to 1, but in this example j is not the next ideal after i in lex order. The key point is that all ideals between i and j in lex order agree with i and j on all bits higher order than b . Since j has rank $m + 1$ and all bits from 0 through b of j are 1, each ideal i_d , $0 \leq d \leq q$, has exactly one 0 bit in positions 0 through b . Furthermore, the position of the 0 bit moves left to right down the list as shown above.

Now the ideal generation algorithm would go from i_d to i_{d+1} , potentially through some intermediate ideals, changing the setting of some bits to the right of b_d as necessary. Eventually, all of these bits become reset to 1 (giving j), then bit b_{d+1} is set to 0, and the process continues in this fashion, generating each i_d in turn and eventually j .

This suggests a way to bound the ideals considered from below. Start the algorithm with an initial ideal whose m lowest order bits are 1 and all other bits are 0. By Corollary 6.2, this is the smallest ideal of interest in lex order. The most recently generated ideal of interest is used to find the next ideal of interest in the loop beginning at

step 5. Then, at step 5.2, ideal j is found with rank greater than m . Note that the loop beginning at step 5.3 causes the rank of j to decrease. Process this loop as before, but terminate this loop if the rank of j becomes m . The algorithm thus does not process ideals of rank smaller than m .

In the example above, the ideals are generated in this order, with no intervening ideals of rank less than m processed:

- $i_0 = i$ bit $b_0 = b$ is 0
- i_1 bit $b_1 < b_0$ is 0
- .
- .
- .
- i_q bit $b_q < b_{q-1}$ is 0
- j

This bounds the processing of ideals from below. It remains to bound the processing of ideals from above. This will be accomplished by defining the largest ideal of interest in lex order.

Given two ideals i and j with the same rank, the greater ideal in lex order is the one with the highest-order 1 bit. It is useful to consider an example to see the pattern used to define the maximum ideal of a given rank. Let $k = 3$ and $r = 4$; i.e., consider all ideals of rank 4 in $\mathcal{I}(B_3)$. These ideals are listed in lex order Table 6.1.

Table 6.1. Ideals of Rank 4 in $\mathcal{I}(B_3)$ in Lex Order

Ideal Characteristic Vector	Bit Positions Set
00001111	0, 1, 2, 3
00010111	0, 1, 2, 4
00110011	0, 1, 4, 5
01010101	0, 2, 4, 6

Definition 6.8: The *height* of a vertex in B_k is its distance from the bottom vertex.

The *height* of an ideal i in $\mathcal{J}(B_k)$ is the maximum height of a vertex in i .

Note that the height of a vertex in a Boolean lattice is the number of 1 bits in its bit representation.

Inspection of Table 6.1 shows that the highest bit position set in any ideal in the table is bit 6. A key point is that this bit position has a maximum height of 2. Two other ideals have a bit position of height 2 set. Note that the highest bit position (6) set in any ideal in the table is the join of the atoms with the two highest-order atom bit positions.

This suggests the following greedy heuristic for finding the maximum ideal in lex order of a specified rank:

- use the highest-order atom not previously used when a new atom is needed
- build the ideal to maximize its height (the maximum height of a vertex in the ideal)

This is the basis for an algorithm that, in fact, generates the maximum ideals for all the ranks of $\mathcal{J}(B_k)$. For $\mathcal{J}(B_3)$, the bit positions used to generate the maximum ideals of a given rank are selected in the order shown in Table 6.2:

Table 6.2. Bit Position Order for Generating Max Ideals in $\mathcal{J}(B_3)$

Rank	Next Bit Position for Maximum Ideal	Maximum Ideal of Rank in $\mathcal{J}(B_3)$
0		00000000
1	0	00000001
2	4	00010001
3	2	00010101
4	6	01010101
5	1	01010111
6	5	01110111
7	3	01111111
8	7	11111111

Bit position 0 must be picked in any nonempty ideal. The next bit picked is bit position 4, the highest-order atom position; the set of bits $\{0, 4\}$ defines the maximum ideal of rank 2. Then pick the next highest order atom position, 2, to give the maximum ideal of rank 3. We are now prepared to pick the highest-order bit position (6) with height 2. Then the atom at bit position 1 is selected and merged in succeeding steps to form the highest possible composites.

A pattern clearly emerges. The first entry $p(0)$ is null and the second entry $p(1)$ is 0 (which is in every nonempty ideal). Then repeatedly select the highest remaining atom in bit-position order and add its value to the previous bit positions in order. In this example, the first atom has bit position 4, so add 4 to 0 to get 4. The next atom has bit position 2, so add 2 to 0 and 4 to get (respectively) 2 and 6. Then atom 1 is added to 0, 4, 2, and 6 to give 1, 5, 3, and 7. The process terminates when all atoms have been used.

The max ideal generation algorithm given below calculates the order of bits used to generate maximum ideals and returns the maximum ideal of a given rank r .

ALGORITHM: Max Ideal Generation Algorithm

INPUT: $k \geq 0$, the number of atoms in B_k , and rank r

OUTPUT: i_{max} , the maximum ideal in lex order of rank r in $\mathcal{J}(B_k)$

- 1 Set $p[0]$ to null.
- 2 Set $p[1] = 0$.
- 3 For each atom a from 1 through k :
 - 3.1 Set $b = 2^{k-a}$; b is the bit position corresponding to the a^{th} highest-order atom.
 - 3.2 For each i from 1 through 2^{a-1} , set $p[2^{a-1}+i] = p[i] + b$.
- 4 Set $i_{max} = 0$.
- 5 For i from 1 through r , set bit $p[i]$ in i_{max} to 1.

The max ideal generation algorithm will now be proved correct. In the following, the bit position of a characteristic vector of an ideal in $\mathcal{J}(B_k)$ is the vertex number of the

vertex in B_k . For example, in $\mathcal{J}(B_3)$, bit positions 0, 1, 2, 3, 4, 5, 6, and 7 correspond to the vertices 000, 001, 010, 011, 100, 101, 110, and 111 in B_3 .

Proposition 6.9: Let $A = \{a_1, \dots, a_k\}$ be the atoms of B_k . Let v be a vertex of B_k in an ideal i of $\mathcal{J}(B_k)$, such that v is greater than or equal to exactly d atoms of A . Let h_i be the height of i . Then

- (a) the bit position of v in the characteristic vector of i is the sum of the bit positions of the d atoms less than or equal to v in B_k ; and
- (b) $d = h_v$, the height of v (and so the height of a vertex in B_k is the number of atoms that generate it);
- (c) $|i| \geq 2^{h_i}$; and
- (d) each atom has a bit position greater than the sum of the bit positions of all atoms less than it in lex order.

Proof: (a) follows from the fact that every vertex v in B_k satisfies $v = \bigvee_{a \in A, a \leq v} a =$

$\bigoplus_{a \in A, a \leq v} a$. (b) follows from the fact that the principal ideal $\downarrow_{\{v\}}$ generated by v is a

Boolean sublattice with d atoms. For (c), let w have maximum height h_i in i . Since i is

an ideal, all 2^{h_i} vertices of the principal ideal $\downarrow_{\{w\}}$ are in i , so $|i| \geq 2^{h_i}$. (d) follows from the

fact that the n^{th} atom from the bottom in lex order has bit position $2^{n-1} = 1 + \sum_{i=0}^{n-2} 2^i$.

Lemma 6.10: Let $2^r \leq n < 2^{r+1}$. Then the largest ideal in lex order of size n contains the principal ideal $\downarrow_{\{v\}}$ generated by v , the join of the r largest atoms in lex order.

Proof: By Proposition 6.9 (c), the height of the largest element of an ideal of cardinality n is at most r . By Proposition 6.9 (a), the largest vertex of height r is generated by the largest sum of at most r atoms, which is (v) .

Clearly, an ideal of cardinality n exists that contains (v) . Just start with (v) and successively add a minimal element from the complement $n - 2^r$ times. Thus, there is an ideal of cardinality n that has the highest possible bit position set (the sum of the largest r atoms). Any ideal i of cardinality n that does not contain (v) does not have this bit position, or any higher bit position, set to 1 in i . This completes the proof.

Lemma 6.11: Let i be the largest ideal in lex order of size n , where $2^r < n < 2^{r+1}$. Then i only uses the $r + 1$ largest atoms of B_k in lex order; i.e., every vertex in i is the join of 0 or more atoms from $\{a_1, \dots, a_{r+1}\}$, the largest $r + 1$ atoms of B_k .

Proof: Let j be an ideal of size n with $2^r < n < 2^{r+1}$ that contains a vertex $v > a$, where a is not one of the largest $r + 1$ atoms of B_k . Without loss of generality, v may be assumed to be maximal in j . Partition the atoms of v into $L + S$, where the atoms of L are in $A_{r+1} = \{a_1, \dots, a_{r+1}\}$ and the atoms of $S \neq \emptyset$ are not in A_{r+1} . There is at least one maximal vertex w in j that contains L and perhaps some other atoms in A_{r+1} but no atoms in A_{r+1}' . (L itself must be in j by the downset condition.) But w does not contain all atoms in A_{r+1} ; if it did, then $|j| \geq 2^{r+1}$ by Proposition 6.9 (c), a contradiction. Then there is a minimal element x in the complement of j that contains L and at least one additional atom b in A_{r+1} and no atoms in A_{r+1}' . Form ideal d from j by removing v and adding x . By Proposition 6.9 (d), the position of x is at least $b + L > S + L$, so $d > j$ in lex order. Repeat this process until all ideals contain only atoms from A_{r+1} .

Lemma 6.12: Let $A = \{a_1, \dots, a_k\}$ be the atoms of B_k in reverse lex order (largest to smallest) and suppose that, for $1 \leq j \leq 2^r$, $\{v_1, \dots, v_j\}$ is the maximum ideal in lex order of rank j . Then, for $1 \leq j \leq 2^r$, $\{v_1, \dots, v_{2^r}, v_1 + a_{r+1}, \dots, v_j + a_{r+1}\}$ is the maximum ideal in lex order of rank $2^r + j$.

Proof: Suppose the lemma is false. Let j be an index such that the maximum ideal m in lex order of rank $2^r + j$ is not $\{v_1, \dots, v_{2^r}, v_1 + a_{r+1}, \dots, v_j + a_{r+1}\}$. By Lemmas 6.10 and 6.11, m contains the $r + 1$ largest atoms and no other atoms. By Lemma 6.10, the vertex $v = \bigvee_{i=1}^r a_i$ is in m . By Lemma 6.11, every vertex v_i in $m \setminus \downarrow_{\{v\}}$ is the join of a_{r+1} and some subset of $\{a_1, \dots, a_r\}$. That is, $m = (v) + \{v_{2^r+1}, \dots, v_{2^r+j}\}$ and each vertex v_i in $\{v_{2^r+1}, \dots, v_{2^r+j}\}$ is the join of a_{r+1} and some subset of $\{a_1, \dots, a_r\}$.

Since $(v) + \{v_{2^r+1}, \dots, v_{2^r+j}\}$ is greater than $\{v_1, \dots, v_{2^r}, v_1 + a_{r+1}, \dots, v_j + a_{r+1}\}$ in lex order, the bit string formed by setting $\{v_{2^r+1}, \dots, v_{2^r+j}\}$ to 1 and all other bits to 0 is greater in lex order than the bit string formed by setting bits $\{v_1 + a_{r+1}, \dots, v_j + a_{r+1}\}$ to 1 and all other bits to 0. Let $w_i = v_{2^r+i} \setminus \{a_{r+1}\}$, $1 \leq i \leq j$. Then the bit string formed by setting bits $\{w_1, \dots, w_j\}$ to 1 and all other bits to 0 is greater in lex order than the bit string formed by setting bits $\{v_1, \dots, v_j\}$ to 1 and all other bits to 0.

Furthermore, $\{w_1, \dots, w_j\}$ form an ideal. For suppose $u < w_i$ for some i , $1 \leq i \leq j$. Then the atoms subsumed by u do not include a_{r+1} and so $u + \{a_{r+1}\} < w_i + \{a_{r+1}\} = v_{2^r+i}$. Since m is an ideal containing v_{2^r+i} , m must contain $u + \{a_{r+1}\}$; i.e., for some q , $u + \{a_{r+1}\} = v_{2^r+q}$. Then $u = v_{2^r+q} \setminus \{a_{r+1}\} = w_q \in \{w_1, \dots, w_j\}$.

But then $\{v_1, \dots, v_j\}$ is not the largest ideal in lex order of cardinality j , a contradiction. This completes the proof.

The hypothesis of Lemma 6.12 is that the set of maximum ideals in lex order for ranks $0, 1, \dots, 2^r$ are nested. That is, the vertices in the maximum ideal of rank $n + 1$ are obtained from the vertices of the maximum ideal of rank n by the addition of one vertex of B_k . This is a very nice property that leads to the greedy max ideal generation algorithm given earlier. Lemma 6.12 assures that, if it is true initially, then it is true for all ranks $0, \dots, 2^k$ of $\mathcal{J}(B_k)$.

Theorem 6.13: The max ideal generation algorithm is correct; that is, it terminates after a finite number of steps with the maximum ideal in lex order of rank r .

Proof: By induction on r .

Basis: For $r = 0$, the only ideal is the null ideal. This is step 1 of the max ideal generation algorithm. For $r = 1$, the only ideal is $\{0\}$, the bottom vertex of B_k . This is step 2 of the max ideal generation algorithm. For $r = 2$, there are $|A|$ ideals $\{0, a\}$ for each atom $a \in A$. The maximum ideal of cardinality 2 is thus $\{0, a_{max}\}$, where a_{max} is the largest atom in lex order. Note that $\{\emptyset, 0, 0 + a_{max}\}$ define nested solutions for $0 \leq r \leq 2$. Also note that, for $j = 0$ atoms, the nested solution set has the $2^j = 1$ vertex $\{0\}$, and for $j = 1$, the nested solution set has the $2^j = 2$ vertices $\{0, a_{max}\}$ obtained by adding a_{max} to the $2^{j-1} = 1$ vertex $\{0\}$. This is step 3.2 in the max ideal generation algorithm for the first (largest) atom.

Induction: Suppose the first 2^n maximum ideals are nested, being the initial segments of $\{p[1] \subseteq \dots \subseteq p[2^n]\}$. By Lemma 6.12, the first $2^n + j$ maximum ideals have

the initial segments of $\{p[1] \subseteq \dots \subseteq p[2^n] \subseteq p[2^n + 1] \subseteq \dots \subseteq p[2^n + j]\}$ as nested solutions for $1 \leq j \leq 2^n$. This is implemented in steps 3, 3.1 and 3.2 of the max ideal generation algorithm. This extends the induction.

Since the only loops are in steps 3, 3.2 and 5 and these loops are all bounded, the algorithm terminates in a finite number of steps. This completes the proof.

Corollary 6.14: The max ideal generation algorithm is greedy and produces a nested family of lexicographically maximum ideals, one for each rank in $\mathcal{J}(P)$.

Proof: Steps 4 and 5 of the max ideal generation algorithm implement the nested solutions of Theorem 6.13 for given r .

It is worth noting that the maximum ideals in lex order of ranks $2^r + 1, \dots, 2^{r+1}$ are formed, in order, from the maximum ideals in lex order of ranks $0, \dots, 2^r$ as the products with the chain $0 - a_{r+1}$. This product construction is shown in Figure 6.1 for B_4 . The numbers on the edges show the order in which vertices are added to form the next max ideal via “product edges” from B_{r-1} to B_r using the largest remaining atom.

The main differences, then, between the selective ideal generation algorithm and the ideal generation algorithm are the specification of ideal ranks of interest, generation of minimum and maximum ideals of interest, and the behavior of the loop that sets bits to 0 in j . The selective ideal generation algorithm follows. The input value i_{max} is obtained using the max ideal generation algorithm.

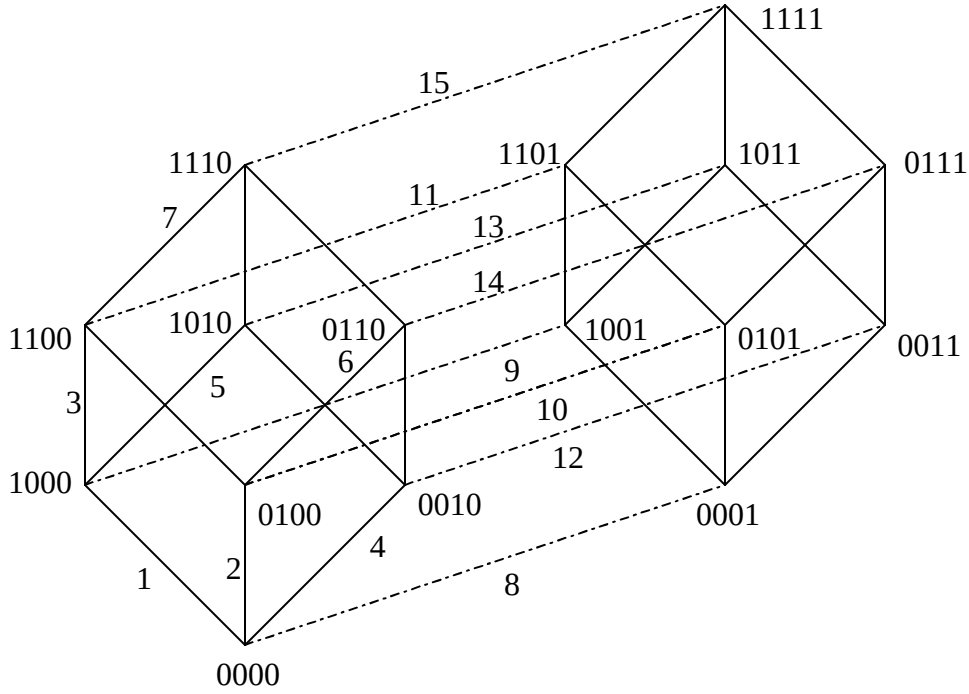


Figure 6.1. Product Construction of B_4 from B_3

ALGORITHM: Selective Ideal Generation Algorithm

INPUT: Finite poset $(P, \leq)$, set R of ranks of interest, $i_{\max} = \max$ ideal of interest

OUTPUT: Array $I[r]$ of ideals of P in lex order, Whitney number $W[r]$ for all r in R

- 1 Define a total extension $f: P \rightarrow \mathcal{N}_{|P|}$ so that, if $u \leq v$ in P , then $f(u) \leq f(v)$ in $\mathcal{N}_{|P|}$.
- 2 For each i in $\mathcal{N}_{|P|}$, construct a list $L[i]$ of successors j such that $f^{-1}(i) < \bullet f^{-1}(j)$ in P .
- 3 For each rank r from 0 through $2^{|P|}$:
 - 3.1 Set $g[r]$ TRUE to generate all ideals of rank r .
 - 3.2 Set $g[r]$ FALSE to skip all ideals of rank r .
- 4 Find m , the smallest r for which $g[r]$ is TRUE; terminate if there is no such r .
- 5 Initialize $i = I[m][0] = 2^m - 1$, $r = m$, $W[m] = 1$, $W[s] = 0$ for all $s \neq r$, $0 \leq s \leq 2^{|P|}$
- 6 While the most recently generated ideal $i < i_{\max}$:
 - 6.1 Find the lowest order 0 bit b in i , the most recently generated ideal.
 - 6.2 Set j to the result of setting bit b of i to 1, leaving all other bits the same.
 - 6.3 Increment r , which is now the rank of j . Note that $r > m$.
 - 6.4 For each bit d in j to the right of b , moving left to right from $b - 1$ to 0,
 - 6.4.1 If no member of $L(d)$ has its bit set to 1 in j ,
 - 6.4.1.1 Set bit d to 0 in j .
 - 6.4.1.2 Decrement r . Note that $r \geq m$.
 - 6.4.1.3 If $r = m$, exit loop 6.4.
 - 6.5 If $g[r]$ is true, set $i = I[r][W[r]] = j$ and increment $W[r]$
- 7 End of the while loop.

In the following, we will refer to an ideal whose rank r satisfies $g[r] = \text{true}$ as an “ideal of interest”, and r as a “rank of interest”. Note that the ranks of interest need not be contiguous. Also note that this algorithm works for any ranked poset. Finally, note that the algorithm halts when the max ideal in lex order of the maximum rank of interest is generated. This is a very significant performance enhancement; without it, all ideals larger in lex order than the minimum ideal of interest would need to be generated.

We now prove the selective ideal generation algorithm correct. The two main points to prove about the selective ideal generation algorithm are that all ideals of rank m are found in the right order in loop 6.4, and that all ideals of interest with rank greater than m are found in the right order.

Lemma 6.15: Given a finite poset P , the selective ideal generation algorithm terminates in a finite number of steps.

Proof: All loops are bounded as in the ideal generation algorithm. The selective ideal generation algorithm will terminate in fewer steps than the ideal generation algorithm unless all ideals are selected to be of interest.

Lemma 6.16: The selective ideal generation algorithm generates only ideals of interest.

Proof: An ideal of rank m , the smallest rank of interest, is generated in step 5. All other ideals are generated in step 6.5, and only if they are of interest.

Lemma 6.17: Let $m < |P|$, let i be an ideal of rank m generated by the selective ideal generation algorithm, and let j be the first ideal after i in lex order such that $r(j) > m$. Suppose further that there are precisely q ideals $i_1, \dots, i_q$ of rank m that fall between i and

j in the lex ordering, with $i < i_1 < \dots < i_q < j$. Then the selective ideal generation algorithm generates ideals $i, i_1, \dots, i_q$ consecutively in lex order, and the next ideal generated by the selective ideal generation algorithm after i_q is the smallest ideal u of interest such that $u \geq j$ in lex order.

Proof: By Lemma 6.1, j is the result of setting the least significant 0 bit b of i to 1 and leaving all other bits of i unchanged. Since $i < i_1 < \dots < i_q < j$, all of these ideals agree at all bit positions higher order than b . Since $r(j) = m + 1$ and j has a 1 in all bit positions 0 through b , each of $i, i_1, \dots, i_q$ has exactly one 0 bit and b 1 bits in bit positions 0 through b . For i , the 0 bit is in position b . Since $i < i_1 < \dots < i_q < j$, it follows that the 0 bit positions must satisfy $b > b_1 > \dots > b_q$, where b_d is the lowest order 0 bit in ideal i_d .

Inspection of the selective ideal generation algorithm shows that the next ideal generated after i is obtained by setting bit b to 1 in step 6.2 and bit b_1 to 0 in step 6.4.1.1. (Bit b_1 must pass the test at step 6.4.1, since i_1 is an ideal. If any bit between b and b_1 passed this test, the result of changing that bit from 1 to 0 would be an ideal of rank m between i and i_1 , a contradiction.)

Suppose $d < q$ and ideals $i, i_1, \dots, i_d$ are generated in lex order. By the same argument, the next ideal generated after i_d by the selective ideal generation algorithm is i_{d+1} . Thus all ideals $i, i_1, \dots, i_q$ are generated in lex order. By Lemma 6.1, the next ideal processed by the selective ideal generation algorithm is j . If $m + 1 = r(j)$ is a rank of interest, then the selective ideal generation algorithm generates j next after i_d ; otherwise, the next ideal generated after i_d is the smallest ideal u of interest such that $u \geq j$ in lex order.

Corollary 6.18: The selective ideal generation algorithm generates all ideals of rank m in lex order.

Proof: Partition the ideals of rank m as follows, where each ideal i_p has rank m and each ideal j_q has rank $m + 1$: $2^m - 1 = i_1 < \dots < i_{p_1} < j_1 < \dots < i_{s+1} < \dots < i_{s+p_t} < j_t$, where the $\{j_q\}$ are all the ideals of rank $m + 1$ that occur in lex order between the smallest and largest ideals of rank m in lex order, the partitions are separated by the $\{j_q\}$, and $s = \sum_{n \in \{1, \dots, t-1\}} p_n$. The result follows by applying Lemma 6.17 to each of the t blocks of ideals of rank $m + 1$.

Lemma 6.19: Every ideal of interest is generated by the selective ideal generation algorithm. Furthermore, the ideals generated by the selective ideal generation algorithm are generated in lex order.

Proof: By Corollary 6.2, the first ideal of interest is $2^m - 1$, which has the m lowest-order bits set to 1 and all other bits set to 0. This is the first ideal generated, in step 5. By Corollary 6.18, all ideals of rank m are generated in the same relative lex order as in the ideal generation algorithm. By step 6.4.1.3, an ideal is not processed only if its rank is smaller than m . Thus all ideals of rank larger than m are processed in the same order as in the ideal generation algorithm, and each such ideal is generated in step 6.5 iff it is an ideal of interest.

Theorem 6.20: The selective ideal generation algorithm is correct. That is,

a) The selective ideal generation algorithm terminates in a finite number of steps, and

- b) Upon completion of the selective ideal generation algorithm, the ideals generated are precisely the ideals of interest. Moreover,
- c) The ideals are generated in lex order. They are stored by rank, and by lex order within each rank.

Proof:

- a) Lemma 6.15.
- b) Lemma 6.16 and Lemma 6.19.
- c) Note that, in step 6.5, the ideals are stored by rank as they are generated and determined to be of interest. By Corollary 6.18, the ideals are generated in lex order, and thus are stored by lex order within each rank. This completes the proof.

For each rank r of interest, the value $W[r]$ at the end of the selective ideal generation algorithm is the Whitney number of rank r .

We conclude this chapter with a result that significantly reduces the computation necessary to form the quotient poset. As noted in Chapter 5, the vertices of the quotient poset are the equivalence classes of the vertices of the ideal poset. When a new ideal i of interest is generated, i must be compared with existing ideals to determine if i is a new class representative or an additional member of an existing class. Furthermore, the edges coming into the equivalence class of i must be updated to reflect any edges coming into i specifically. We consider the implications of each of these steps in turn.

Note that the class representatives are the first ideals encountered in lex order. A brute-force approach to testing a new ideal i for membership in an existing equivalence class would be to perform all permutations of i and compare each to the class

representatives of ideals of the same rank that have already been found. This involves checking all permutations of i against all class representatives of the same rank as i until either a match has been found or no more permutations remain. This requires $k!$ permutations of i and comparison of each permutation with the representative for each class. The only way to exit this exhaustive testing process early is if an isomorphism test passes. But isomorphism tests usually fail, so the full gamut of permutations and tests must be run most of the time.

Fortunately, there is an easily-implemented test that causes early return from an isomorphism test that has no chance of passing. Let us define the *profile* of an ideal i as the map $profile : (i, r) \rightarrow c$, where c is the number of members of i that are of rank r in B_k . Obviously, if $i \sim j$, then $profile(i, r) = profile(j, r)$ for all r . This test will frequently fail, providing an early exit from the isomorphism test process. Note that the profile of an ideal can easily be calculated and stored with the equivalence class when it is generated.

An efficient algorithm to determine the edges in the quotient poset network follows from some observations about the action of S_k on $\mathcal{J}(B_k)$. At first glance, when a new ideal i is found, it would appear necessary to test for $[j] < \bullet [i]$ by generating all permutations of i and/or j , testing each combination for coverage. Because the ideals are generated in lex order, this is not necessary. The proof involves some intermediate lemmas, which follow. Note that a new ideal is only used once in the coverage test. Thus, if $j < \bullet i$ but j is not the class representative, i might not cover the class representative, so we have to make sure that (an edge isomorphic to) this edge is picked up in some other test.

Lemma 6.21: If $j <_{\bullet} i$, then i occurs after j in lex order.

Proof: If $j <_{\bullet} i$, Every bit position in j that is a 1 must also be a 1 in i , and i must have exactly one additional 1 bit. Thus $i > j$ in numerical (lex) order.

Lemma 6.22: If i is an ideal of B_k and p is a permutation over the k atoms of B_k , then $p(i)$ is an ideal.

Proof: Let i be an ideal of B_k and let p be a permutation over the k atoms of B_k . Let $x \in p(i)$ and let $y \leq x$. Then the bit representation of x as an element of B_k has a 1 in every position where there is a 1 in the bit representation of y as an element of B_k . Then $p^{-1}(x)$ has a 1 in every bit position where there is a 1 in $p^{-1}(y)$; i.e., $p^{-1}(y) \leq p^{-1}(x)$. But since $x \in p(i)$, $p^{-1}(x) \in i$, and since i is an ideal, $p^{-1}(y) \in i$, and so $y \in p(i)$. Therefore $p(i)$ is an ideal.

Lemma 6.23: If p is a permutation over the k atoms of B_k and i and j are ideals of B_k such that $j <_{\bullet} i$, then $p(j) <_{\bullet} p(i)$.

Proof: Since a permutation p simply reorders the bits in a characteristic vector, p is rank-preserving. Let $j <_{\bullet} i$ and suppose bit position b has the same value (0 or 1) in i and j . Then bit position $p(b)$ has this same value in $p(i)$ and $p(j)$. Since $j <_{\bullet} i$, this accounts for all positions except 1, say bit position c , where j has a 0 and i has a 1. Then bit position $p(c)$ is 0 in $p(j)$ and 1 in $p(i)$. Therefore $p(j) <_{\bullet} p(i)$.

Corollary 6.24: A permutation p on the k atoms of B_k induces an automorphism on $\mathcal{J}(B_k)$.

Proof: From Lemma 6.23 and the observation that permutations are invertible, if $p(j) <_{\bullet} p(i)$, then $j = p^{-1}p(j) <_{\bullet} p^{-1}p(i) = i$. Therefore $j <_{\bullet} i$ iff $p(j) <_{\bullet} p(i)$.

Theorem 6.25: Suppose j is the class representative of $[j]$, $a \sim j$, and $a <_{\bullet} b$. Then there exists an ideal i such that $i \sim b$, $j <_{\bullet} i$, and i occurs after j in lex order.

Proof: Since $a \sim j$, $p(a) = j$ for some permutation p on the k atoms of B_k . Let $i = p(b)$; then $i \sim b$. By Lemma 6.23, since $a <_{\bullet} b$, $j = p(a) <_{\bullet} p(b) = i$. By Lemma 6.21, i occurs after j in lex order.

Corollary 6.26: Let the ideals of B_k be generated in lex order. To generate the edges in the ideal quotient network, it suffices to test each ideal generated for coverage of class representatives of ideals of the previous rank (without permuting the previously generated ideals).

Proof: An edge is needed from $[j]$ to $[m]$ iff $a <_{\bullet} b$ for some $a \sim j$ and $b \sim m$. By Theorem 6.25, this is true iff $j <_{\bullet} i$, where $p(a) = j$ and $p(b) = i$. But then $i \sim b \sim m$ and i occurs after j in lex order. The necessary edge then results from the test on $j <_{\bullet} i$, where j is the class representative and (because i occurs after j in lex order) j already exists in the network.

Thus isomorphism tests are needed only for updating the vertices, not the edges, of the ideal quotient network.

CHAPTER 7: THE IDEAL EXCHANGE ALGORITHM

As pointed out in Chapters 5 and 6, the utility of generating equivalence classes of an ideal for a selected range of interest is that this allows for normality testing to occur in parallel. This led to the selective ideal generation algorithm, a modification of the traditional ideal generation algorithm, presented in Chapter 6.

This yields a very significant time savings, as the normality testing algorithm can take several days to find a normal flow between two adjacent ranks of $\mathcal{J}(B_6)$. The question naturally arises as to whether the algorithm can be further improved by changing the computational structure.

This chapter answers that question in the affirmative by developing an algorithm based directly on the ideal exchange graph. Let us suppose that we have an ideal i of rank (cardinality) r . Using the selective ideal generation algorithm, the next ideal in lex order of rank r might not be found until several intervening ideals of rank different from r have been processed. The philosophy of the new algorithm instead examines the ideal exchange graph to determine an optimal way to swap an equal number of vertices of B_k (or any ranked poset) between an ideal i and its complement $B_k - i$. For this reason, the new algorithm is called the “ideal exchange algorithm”.

Comparative execution times for cases analyzed in this study using the selective ideal generation algorithm and the ideal exchange algorithm are given in Chapter 9. The ideal exchange algorithm is defined and proved correct in this chapter.

We start with an example to motivate its development. The example is based on the ideals of rank 8 in B_4 , which are shown in lex order in Table 7.1 below.

Table 7.1. Ideals of Rank 8 in Lex Order in $\mathcal{I}(B_4)$

n	n^{th} ideal of rank 8	Max vertices	Min complement	Exchange
1	0000000011111111	7	8	
2	0000000101111111	3, 5, 6, 8	7, 9, 10, 12	$7 \rightarrow 8$
3	0000001100111111	3, 5, 9	6, 10, 12	$6 \rightarrow 9$
4	0000001101011111	3, 6, 9	5, 10, 12	$5 \rightarrow 6$
5	0000001101110111	5, 6, 9	3, 10, 12	$3 \rightarrow 5$
6	0000010100111111	3, 5, 10	6, 9, 12	$6, 9 \rightarrow 3, 10$
7	0000010101011111	3, 6, 10	5, 9, 12	$5 \rightarrow 6$
8	0000010101110111	5, 6, 10	3, 9, 12	$3 \rightarrow 5$
9	0000011100011111	3, 4, 9, 10	5, 6, 11, 12	$5, 6 \rightarrow 3, 9$
10	0000011100110111	5, 9, 10	3, 6, 12	$3 \rightarrow 5$
11	0000011101010111	6, 9, 10	3, 5, 12	$5 \rightarrow 6$
12	0000111100001111	11	4	$4, 6 \rightarrow 3, 11$
13	0001000100111111	3, 5, 12	6, 9, 10	$9, 10, 11 \rightarrow 4, 5, 12$
14	0001000101011111	3, 6, 12	5, 9, 10	$5 \rightarrow 6$
15	0001000101110111	5, 6, 12	3, 9, 10	$3 \rightarrow 5$
16	0001001100011111	3, 9, 12	5, 6, 10	$5, 6 \rightarrow 3, 9$
17	0001001100110111	2, 5, 9, 12	3, 6, 10, 13	$3 \rightarrow 5$
18	0001001101010111	6, 9, 12	3, 5, 10	$5 \rightarrow 6$
19	0001010100011111	3, 10, 12	5, 6, 9	$6, 9 \rightarrow 3, 10$
20	0001010100110111	5, 10, 12	3, 6, 9	$3 \rightarrow 5$
21	0001010101010111	1, 6, 10, 12	3, 5, 9, 14	$5 \rightarrow 6$
22	0001011100010111	9, 10, 12	3, 5, 6	$6 \rightarrow 9$
23	0011001100110011	13	2	$2, 10 \rightarrow 5, 13$
24	0101010101010101	14	1	$1, 5, 9, 13 \rightarrow 2, 6, 10, 14$

The first ideal has the 8 lowest-order bits set to 1 and the 8 highest-order bits set to 0, in accordance with Corollary 6.2. To obtain the next ideal of rank 8 in lex order, we start with the first ideal and set bit 8 to 1 and bit 7 to 0. This is an ideal because we have exchanged a maximal vertex (7) in the first ideal with a minimal vertex (8) in its complement, and 8 does not cover 7. (If 8 covered 7, this exchange would violate the

downset condition.) The new ideal clearly has cardinality 8, and it is the next ideal in lex order because no bit string with 8 bits set to 1 lies in between the two ideals.

The second entry in Table 7.1 shows the resulting ideal, how it was obtained from the previous ideal (7 is replaced by 8), and the new set of “boundary vertices” (i.e., maximal vertices in the ideal and minimal vertices in the complement). The boundary vertices are obviously key to any approach using the ideal exchange graph to generate ideals.

Our goal is to build an algorithm that generates the ideals of a given rank in lex order. To better understand the method, consider how to generate the third ideal of rank 8 in lex order. Let us look for the smallest 0 vertex that can be exchanged for a smaller 1 vertex. Vertex 7 cannot be chosen because vertex 7 subsumes all smaller vertices in B_4 . This is apparent because the bit representation of 7 has a 1 bit in every position in which a smaller vertex position has a 1 bit. The next smallest 0 bit is in position 9. We look for a smaller 1 bit to convert to 0. Clearly, it is to our advantage to pick this position as high as possible, but not exceeding bit position 9 (because the result needs to be a larger binary integer). The highest such bit position is 8, so we might be tempted to exchange vertices 8 and 9. However, 9 covers 8, so this exchange would violate the downset condition. Since 8 won’t work, we try 6 and see that 6 and 9 can be exchanged without violating the downset condition.

Note that we searched for the next smaller 1 bit below 8 rather than the next higher 1 bit above 9. This is because we want to have the highest-order 1 bit to be as low as possible in order to preserve the lex ordering of the generated ideals.

Ideals 4 and 5 are generated in like manner. Now consider constructing ideal 6 from ideal 5. As before, we choose to swap 9 out and 10 in. This would give us 0000010101110111, which is entry 8 in Table 7.1. The only positions where ideal entries 6 and 8 differ are 3 and 6. We realize that we can exchange vertices 3 and 6, resulting in a smaller ideal in lex order. In general, pick the largest 1 bit position b smaller than any 1 bit positions already used in processing this ideal (because we have already considered all the larger positions). Swap the vertex in this position with the smallest 0 bit position u as long as u is smaller than b (else the resulting ideal is larger) and the downset condition is not violated.

Note that the downset condition is preserved iff the 1 bit b swapped is maximal in the ideal and the 0 bit u swapped is minimal in the complement. This is because u is smaller than b and so u does not cover b .

Entries 7 through 11 are constructed in like manner. Entry 12 provides another twist. In order for a vertex to be swapped from the complement into the ideal to produce a new ideal that is greater in lex order, the vertex being swapped in must have a larger bit position than at least one maximal vertex to be swapped out. Thus the first 0 position to consider is 7. As noted earlier, vertex 7 is greater than any vertex in a lower position, so the downset condition would be violated if vertex 7 were swapped in and an earlier vertex were swapped out.

The next lowest 0 vertex position is 11, and the next lowest vertex position that is minimal in the complement is 12. If we restrict ourselves to maximal and minimal elements, we would swap the 10 out and the 12 in. The new maximal ideal vertex set

would be $\{6, 9, 12\}$, and the new minimal complement vertex set would be $\{3, 5, 10\}$.

As for entry 6, we would swap the 9 and 3 entries, then the 6 and 5 entries to give

0001000100111111.

But this is entry 13, not entry 12. To generate the skipped ideal, we must set the lowest 0 bit to 1 that can be extended to an ideal of rank 8, whether or not it is minimal in the complement. This means setting bit 11 to 1. To preserve the downset condition, all positions subsumed by 11 in B_4 must also be set to 1. Since 11 is not minimal in the complement, at least one other bit must be set from 0 to 1. (In this case, 3 is the only bit for which this must be done.) The maximal elements in entry 11 are $\{6, 9, 10\}$ and bit position 11 covers 9 and 10, so the only maximal that can be swapped to the complement is vertex 6. We still need to swap one more vertex. By moving vertices 3 and 11 to the ideal and vertex 6 to the complement, the boundary has changed: the maximal ideal vertices are $\{4, 11\}$ and the minimal complement vertices are $\{7, 12\}$. Vertex 4 can now be moved to the complement to give the ideal in entry 12.

The remainder of Table 7.1 is constructed in this fashion. It is worth analyzing the last entry to understand performance advantages of the ideal exchange algorithm. The smallest vertex position that can be swapped from the complement to the ideal is 14. We must also swap vertices 2, 6, and 10 into the ideal to preserve the downset condition. After doing that, the maximal vertices in the ideal are $\{13, 14\}$ and the minimal vertex in the complement is $\{3\}$. We need to swap four vertices from the ideal to the complement. Swapping 13 from the ideal to the complement gives maximal ideal vertices $\{5, 9, 14\}$ and minimal complement vertices $\{3, 13\}$. Swapping 9 from the ideal to the complement

gives maximal ideal vertices $\{5, 14\}$ and minimal complement vertices $\{3, 9\}$. Swapping 5 from the ideal to the complement gives maximal ideal vertices $\{1, 14\}$ and $\{3, 5, 9\}$. Swapping 1 from the ideal to the complement completes the ideal.

Note that each swap from the ideal to the complement moved the largest maximal less than 14. This is the best we can hope to do. We must stay below bit 14 because bit 14 was just moved into the ideal and all higher-order bits must remain the same to keep the next ideal as small as possible.

Note also that the bit positions set in the last entry of Table 7.1 match the first 8 bits generated by the max ideal generation algorithm (in the order 0, 8, 4, 12, 2, 10, 6, 14) and so, by Theorem 6.13, this is the largest ideal in lex order of rank 8 in $\mathcal{I}(B_4)$. As in the selective ideal generation algorithm, this becomes the termination condition for the ideal exchange algorithm.

Suppose we did not use this as our termination condition and attempted to construct another ideal using the above approach. The first and only 0 vertex that is larger than at least one maximal ideal vertex is 15. If we add vertex 15 to the ideal, we must add every element because 15 is the top vertex in B_4 . This adds 8 vertices to the ideal, so 8 vertices must be removed without removing vertex 15. But this is impossible without violating the downset condition. In this case, adding a vertex that is not minimal to the ideal does not allow the ideal to be extended to a new ideal of rank 8. The question naturally arises as to whether this case only occurs when no more ideals of a given rank exist. The answer is no. It is indeed possible to move a non-minimal vertex and all its predecessors from the complement to the ideal and “block” in the attempt to move the

same number of vertices from the ideal to the complement. If this happens, the next larger 0 bit needs to be set from 0 to 1. This is called a “retry”. Although this does not happen in rank 8 of $\mathcal{J}(B_4)$, it happens occasionally in some ranks of $\mathcal{J}(B_4)$, $\mathcal{J}(B_5)$ and $\mathcal{J}(B_6)$. The frequency of retries for a given k and rank is analyzed in Chapter 9.

One final point should be made about generation of the last entry in Table 7.1. The numeric difference between successive ideals tends to increase moving down the table. In fact, this difference is largest by a significant amount between the last two ideals. The selective ideal generation algorithm must go through a potentially large number of ideals to go from the next-to-last ideal to the last ideal for a given rank. Thus the ideal exchange algorithm is better at handling the worst-case computation between successive ideals in lex order than is the selective ideal generation algorithm.

On the other hand, the selective ideal generation algorithm is a little better over a large subset of ideals of interest than is the ideal exchange algorithm, which has some overhead in managing maximal and minimal vertex sets and occasional retries. As a rule of thumb, the ideal exchange algorithm is slightly better for running in parallel and the selective ideal generation algorithm is slightly better for running serially. Chapter 9 compares the runtime behavior of the selective ideal generation algorithm and the ideal exchange algorithm under various conditions.

The main points of the ideal exchange algorithm are summarized below:

- Start with the minimum ideal in lex order of rank r (the lowest r bits set to 1).
- To make the next exchange, find the smallest 0 vertex u (not necessarily minimal in the complement; see for example entry 12 in Table 7.1) such that there is some maximal ideal vertex that has a lower position than u but is not a predecessor of u . Let b be the maximal ideal vertex in the highest bit position that satisfies this condition.

- Move u and all of its 0 predecessors (as for entry 12 of Table 7.1) from the complement to the ideal
- Move b from the ideal to the complement and adjust the boundary vertices. Repeat this process with a new b (smaller than the previous b) until the same number of vertices have been moved in each direction between the ideal and the complement (as for entry 12 of Table 7.1). If this is not possible, retry with a higher vertex position u .
- Consider only boundary vertices in positions lower than the last value of b used. As done for entry 6 of Table 7.1, repeatedly exchange the highest maximal ideal vertex b and the lowest minimal complement vertex u as long as b has a higher bit position than u . Adjust the boundary after each exchange.
- Terminate when the algorithm generates the max ideal from the max ideal generation algorithm.

Two efficiency enhancements can be made. The first is to pre-compute the height of each vertex of B_k . Then, for example, if ideals of rank 3 are being computed, skip vertex 3 since any ideal containing vertex 3 must have at least the 4 vertices $\{0, 1, 2, 3\}$.

The other efficiency enhancement involves calculation of the boundary vertices. Note that, to create the boundary vertex sets, each of the 2^k vertices of the Boolean lattice B_k must be processed. The number of edges into and out of any vertex is bounded in each direction by the number k of atoms. This is an $n \cdot \log n$ process where $n = 2^k$, the number of vertices. But once the boundary sets have been made, they do not need to be recomputed after an exchange. Instead, adjust the boundary by examining the successors of vertices pulled into the ideal for minimality in the complement, and the predecessors of vertices pushed out of the ideal for maximality in the ideal. This is $\binom{k}{k/2} \cdot \log 2^k$ in the worst case, and this worst case does not happen all the time.

There is one case in which the boundary vertices must be recomputed from scratch. This occurs when a retry is necessary. Some boundary adjustments may occur before this can be determined. To prepare for the eventuality of a retry, use a “shadow”

copy of the most recently generated ideal i . Then restore i and re-compute the boundary vertices if a retry is necessary; otherwise the next ideal in lex order becomes the shadow ideal after it has been fully processed.

We are now ready to give the ideal exchange algorithm. A proof of correctness will follow.

ALGORITHM: Ideal Exchange Algorithm

INPUT: Finite ranked poset $(P, \leq)$, rank r , $i_{\max} = \max$ ideal of rank r

OUTPUT: $I[r]$ = ideals of rank r in lex order, Whitney number $W[r]$, c_r retries.

- 1 Define a total extension $f: P \rightarrow \mathcal{N}_{|P|}$ so that, if $u \leq v$ in P , then $f(u) \leq f(v)$ in $\mathcal{N}_{|P|}$.
- 2 For each x in $\mathcal{N}_{|P|}$, construct a list of the successors y such that $f^{-1}(x) < \bullet f^{-1}(y)$ in P .
- 3 For each x in $\mathcal{N}_{|P|}$, construct a list of the predecessors z such that $f^{-1}(z) < \bullet f^{-1}(x)$ in P .
- 4 For each x in $\mathcal{N}_{|P|}$, construct $H[x]$, the height of x in P .
- 5 Initialize $i = I[r][0] = 2^r - 1$, $W[r] = 1$, $c_r = 0$, $e = \text{false}$, $u_m = 0$.
- 6 While the most recently generated ideal $i < i_{\max}$:
 - 6.1 Set $j = i$; j will “shadow” i until i can be extended to the next lex order ideal.
 - 6.2 If e is false, construct the upper boundary U_b and lower boundary L_b of i .
 - 6.3 Find the smallest 0 bit u such that all of the following conditions hold:
 - 6.3.1 $u \geq u_m$
 - 6.3.2 $H[u] \leq 2^r$
 - 6.3.3 u is larger than a maximal 1 bit m such that $f^{-1}(m) < f^{-1}(u)$ is false.
 - 6.4 Set b to the largest bit position m satisfying 6.3.3.
 - 6.5 Move u and all predecessors not in i to j ; let n_p = number of vertices moved.
 - 6.6 Adjust the upper boundary U_b and lower boundary L_b of j .
 - 6.7 Do steps 6.7.1 through 6.7.4 n_p times:
 - 6.7.1 If no maximal vertex of j occurs at bit b or earlier, do the following:
 - 6.7.1.1 Set $e = \text{false}$, set $u_m = u + 1$, and increment c_r .
 - 6.7.1.2 Go to step 6.1.
 - 6.7.2 Find the largest bit position $m \leq b$ of a maximal vertex in j .
 - 6.7.3 Set $b = m$ and move b out of j .
 - 6.7.4 Adjust the upper boundary U_b and lower boundary L_b of j .
 - 6.8 Repeat steps 6.8.1 through 6.8.4:
 - 6.8.1 Find m , the largest maximal in L_b such that $m < b$. If there is no m , go to 6.9.
 - 6.8.2 Set $b = m$.
 - 6.8.3 Let u be the smallest minimal vertex in U_b . If $U_b = \emptyset$ or $u > b$, go to step 6.9.
 - 6.8.4 Move u into j , move b out of j , and adjust the boundaries L_b and U_b .
 - 6.9 Set $i = I[r][W[r]] = j$, increment $W[r]$, set $e = \text{true}$, set $u_m = 0$.
- 7 End of the while loop.

We conclude this chapter by proving that the ideal exchange algorithm is correct. In the following (as in the algorithm), “smaller” and “larger” refer to the numerical bit position ordering in the extension of the partial order of P to the totally ordered segment $\mathcal{N}_{|P|}$, the integers $\{0 < \dots < |P| - 1\}$. For $P = B_k$, this is the ordering given by the numerical order of the integers represented by the characteristic vectors in B_k . When we wish to use the order relation $v < w$ in P , we will say that w *subsumes* v .

Lemma 7.1: Let L_b and U_b be the lower and upper boundary sets of an ideal i of rank r in a ranked poset P . That is, L_b is the set of maximal vertices of i and U_b is the set of minimal vertices in the complement $P - i$. Then

- (a) If $u \neq 1$ is the smallest vertex in U_b , then the smallest vertex v in the new upper boundary resulting from moving u from $P \setminus i$ to i satisfies $u < v$.
- (b) If $b \neq 0$ is the largest vertex in L_b smaller than a fixed vertex u_b , then the largest vertex c in the new lower boundary resulting from moving b from i to $P \setminus i$ satisfies $c < b < u_b$.

Proof:

- (a) The new upper boundary consists of all vertices of $U_b \setminus \{u\}$ together with all successors of u that do not subsume any vertex in $U_b \setminus \{u\}$. By the hypothesis, u is smaller than every vertex in $U_b \setminus \{u\}$, and u is smaller than all of its successors in any total extension of $\leq$ in P to $<$ in the natural numbers.

- (b) Similarly, the new lower boundary consists of all vertices of $L_b \setminus \{b\}$ together with all predecessors of b that are not subsumed by any vertex in $L_b \setminus \{b\}$. By the

hypothesis, b (and thus u_b) is greater than every vertex in $L_b \setminus \{b\}$, and b (and thus u_b) is greater than all of its predecessors.

The fixed vertex u_b does not affect the argument but is technically required for Lemma 7.1 to be used as intended. This also applies to the next lemma.

Lemma 7.2: Let i be an ideal of a ranked poset P . Let $U = \{u_1 < \dots < u_n\}$ be the bit positions of a subset of $P \setminus i$ and $B = \{b_1 < \dots < b_n\}$ be the bit positions of a subset of i , with $|B| = n = |U|$ for some $n > 0$, where U and B satisfy the following conditions:

- (i) $u_n < b_1$,
- (ii) $b_n < u_b$ for a given vertex u_b ,
- (iii) the string of u_b bits with 1 in positions $b_1, \dots, b_n$ and 0 elsewhere is the largest string in lex order of length u_b with n 1 bits such that removal from i of the corresponding n vertices, all from positions less than u_b , is an ideal, and
- (iv) the string of u_b bits with 1 in positions $u_1, \dots, u_n$ and 0 elsewhere is the smallest string in lex order of length u_b with n 1 bits such that addition to i of the corresponding n vertices is an ideal.

Then

- (a) $i \setminus B$, $i + U$, and $(i \setminus B) + U = (i + U) \setminus B$ are all ideals,
- (b) for $0 \leq j < n$, $i + \{u_1, \dots, u_j\}$ is an ideal and u_{j+1} is the smallest minimal vertex bit position less than u_b in $P \setminus (i + \{u_1, \dots, u_j\})$, and
- (c) for $0 \leq m < n$, $i \setminus \{b_{n-m+1}, \dots, b_n\}$ is an ideal and b_{n-m} is the largest maximal vertex bit position less than u_b in $i \setminus \{b_{n-m+1}, \dots, b_n\}$.

Proof:

(a) By (iii), $i \setminus B$ is an ideal. By (iv), $i + U$ is an ideal. Since i and U are disjoint and $B \subseteq i$, $(i \setminus B) + U = (i + U) \setminus B$. Suppose $(i \setminus B) + U$ is not an ideal. Then there exist x and y in P such that $x < y$ in P , $y \in (i \setminus B) + U$, and $x \notin (i \setminus B) + U$. If $y \in (i \setminus B)$, then $x \in (i \setminus B)$ because $(i \setminus B)$ is an ideal. Then y must be in U , and so $y \in i + U$ and (since B and U are disjoint) $y \in (i + U) \setminus B$. Also, $x \notin U$ because $x \notin (i \setminus B) + U$. Since $i + U$ is an ideal and $y \in (i + U)$, $x \in (i + U)$. But then $x \in i$. But $x \notin i \setminus B$, since $x \notin (i \setminus B) + U$. Then $x \in B$, $y \in U$, and $x < y$ in P . But every vertex in B (including x) has a higher bit position than every vertex in U (including y) and the vertex bit position ordering is an extension of the ordering in P , a contradiction.

(b) Note that U may contain some elements that are not minimal in $P \setminus i$. Suppose $i + \{u_1, \dots, u_j\}$ is not an ideal. Then there exist $x \notin i + \{u_1, \dots, u_j\}$ and $y \in i + \{u_1, \dots, u_j\}$ such that $x < y$ in P . Then $x \notin i$, and since i is an ideal, $y \notin i$. Then $y \in \{u_1, \dots, u_j\}$ and $x \notin \{u_1, \dots, u_j\}$. But $x \in U = \{u_1, \dots, u_n\}$ because $i + U$ is an ideal containing y . Then $x = u_p$ for some $p > j$ and $y = u_q$ for some $q \leq j$, and so $u_p > u_q$. This implies $x > y$, a contradiction.

In particular, $i + \{u_1, \dots, u_{n-1}\}$ is an ideal. Clearly, $u_{j+1} \in P \setminus (i + \{u_1, \dots, u_j\})$ and $u_{j+1} < u_p$. Since the vertex numbering is an extension of $\leq$ in P , the smallest vertex in $P \setminus (i + \{u_1, \dots, u_j\})$ is minimal in $P \setminus (i + \{u_1, \dots, u_j\})$. By a similar argument, u_n is maximal in $\{u_1, \dots, u_n\}$. Suppose $w < u_{j+1}$ is the smallest vertex in $P \setminus (i + \{u_1, \dots, u_j\})$. Since $w \in P \setminus (i + \{u_1, \dots, u_j\})$, $w \notin \{u_1, \dots, u_j\}$, and since $w < u_{j+1} < \dots < u_n$, $w \notin \{u_{j+1}, \dots, u_n\}$.

Then $w \notin U$, and therefore $i + \{w, u_1, \dots, u_{n-1}\}$ is an ideal and the bit string of positions $\{w, u_1, \dots, u_{n-1}\}$ is lexicographically smaller than U , a contradiction.

(c) Suppose $i \setminus \{b_{n-m+1}, \dots, b_n\}$ is not an ideal. Then there exist $w < z$ in P such that $w \notin i \setminus \{b_{n-m+1}, \dots, b_n\}$ and $z \in i \setminus \{b_{n-m+1}, \dots, b_n\}$. Then $z \notin \{b_{n-m+1}, \dots, b_n\}$, $z \in i$ and, since i is an ideal, $w \in i$. Since $w \notin i \setminus \{b_{n-m+1}, \dots, b_n\}$, $w \in \{b_{n-m+1}, \dots, b_n\}$, and thus $w \notin i \setminus B$.

Suppose $z \in B = \{b_1, \dots, b_n\}$. Since $z \notin \{b_{n-m+1}, \dots, b_n\}$, $z \in \{b_1, \dots, b_{n-m}\}$. Then $z < w$, since every vertex in $\{b_1, \dots, b_{n-m}\}$ is smaller than every vertex in $\{b_{n-m+1}, \dots, b_n\}$. This is a contradiction; thus $z \notin B$ and so $z \in i / B$. But $i \setminus B$ is an ideal containing z but not w , a contradiction. Therefore $i \setminus \{b_{n-m+1}, \dots, b_n\}$ is an ideal.

In particular, $i \setminus \{b_2, \dots, b_n\}$ is an ideal. Clearly, $b_{n-m} \in i \setminus \{b_{n-m+1}, \dots, b_n\}$, $b_{n-m} < u_b$, and the largest vertex in $i \setminus \{b_{n-m+1}, \dots, b_n\}$ is maximal in $i \setminus \{b_{n-m+1}, \dots, b_n\}$. By a similar argument, b_1 is minimal in $\{b_1, \dots, b_n\}$. Suppose $v > b_{n-m}$ is the largest vertex in $i \setminus \{b_{n-m+1}, \dots, b_n\}$. Since $v \in i \setminus \{b_{n-m+1}, \dots, b_n\}$, $v \notin \{b_{n-m+1}, \dots, b_n\}$, and since $v > b_{n-m} > \dots > b_1$, $v \notin \{b_1, \dots, b_{n-m}\}$. Then $v \notin B$, and therefore $i \setminus \{v, b_2, \dots, b_n\}$ is an ideal and the bit string of positions $\{v, b_2, \dots, b_n\}$ is lexicographically larger than B , a contradiction.

Corollary 7.3: The construction of ideals $i + \{u_1, \dots, u_j\}$ and $i \setminus \{b_{n-m+1}, \dots, b_n\}$ in Lemma 7.2 is greedy. That is, for $0 \leq j < n$, repeatedly pick the smallest vertex not already in the ideal, add it to the ideal and adjust the boundary. For $0 \leq m < n$, repeatedly pick the largest maximal vertex of the ideal in a bit position smaller than u_b , remove it from the ideal and adjust the boundary. Furthermore, these two processes are

independent of each other as long as the vertices added are always in smaller bit positions than the vertices removed.

Proof: Follows from (b) and (c) plus the fact that, since each minimal vertex u_j is smaller than each maximal vertex b_{n-m+1} , no u added can ever be maximal in the ideal and no b removed can ever be minimal in the complement.

Lemma 7.4: Let i be an ideal in a poset P , let $x \in P \setminus i$, and let $Y = \{y \in P \setminus i \mid y \leq x\}$. Then $i + Y$ is the unique smallest ideal of P containing i and x .

Proof: Let $z \in i + Y$ and $w \leq z$ in P . If $z \in i$, then $w \in i$ because i is an ideal. If $z \in Y$, then $z \leq x$. Since $w \leq z$, $w \leq x$, and so either $w \in i$ or $w \in Y$. Therefore $i + Y$ is an ideal. Let j be an ideal containing i and x and let $u \in Y$. Then $u \leq x$, and so $u \in j$. Therefore $i + Y \subseteq j$.

Lemma 7.5: The ideal exchange algorithm only generates ideals of rank r , and they are generated in lex order.

Proof: By induction on the number of ideals generated. By Corollary 6.2, the first ideal generated in step 5 is the smallest ideal of rank r .

Let i be the most recently generated ideal of P . Note that u is used in the ideal exchange algorithm to refer to the bit position of a vertex moved into the ideal. This value changes during the main loop starting at step 6. Let u_b be the bit position found in step 6.3 and sub-steps. By steps 6.5 and 6.9, bit position u_b and all predecessors (a total of n_p elements) not already in i are changed from 0 to 1 in j . By Lemma 7.4, the result is an ideal that has $r + n_p$ elements. By steps 6.3.3 and 6.4, bit b is smaller than bit u_b . In loop 6.7 through 6.7.4, either n_p maximal elements are successively moved out of j (steps

6.7.2 through 6.7.4) or the temporary ideal j is scrapped (step 6.7.1 and sub-steps). By Lemma 7.2 and Corollary 7.3, the result is an ideal that has r elements. In loop 6.8 through 6.8.4, bit position b is moved to successively smaller positions as the corresponding vertices are moved out of j , and the bit positions of vertices moved into j remain smaller than any of the bit positions moved out of j . These exchanges are greedily found in loop 6.8 and made one for one in step 6.8.4, and so by Lemma 7.2 and Corollary 7.3, the result of each exchange is an ideal of rank r . No other vertices are moved into or out of j , and all bits moved out of j are lower-order than u_b , which is changed from 0 to 1. Therefore $i < j$ in lex order and j is assigned to be the next ideal i in step 6.9. This extends the induction.

Lemma 7.6: The ideal exchange algorithm terminates in a finite number of steps for any given P and r .

Proof: For an arbitrary finite ranked poset P , step 1 can be done in polynomial time. One approach is to find the minimal vertices of P , then repeatedly pick any minimal as the next vertex in sequence and recompute the set of minimal vertices. Steps 2 and 3 can be done in polynomial time and are typically available as part of the definition of P . Step 4 is computed for each vertex as the length of any path back to 0. Step 5 is $O(1)$.

Step 6 and sub-steps constitute the main loop of the ideal exchange algorithm. There are finitely many ideals. By Lemma 7.5, each is computed in lex order. The largest ideal i_{max} is given as input and can be computed via the max ideal generation algorithm and Theorem 6.13 for a Boolean lattice. Thus there are finitely many iterations

of the main loop. It suffices to show that each pass through the loop terminates in a finite number of steps.

Step 6.1 is $O(1)$. Step 6.2 computes the maximal vertices of i and the minimal vertices of $P \setminus i$, which can be done in polynomial time. In loop 6.3 through 6.3.3, a vertex u is found (if one exists) satisfying conditions 6.3.1, 6.3.2, and 6.3.3. There are a bounded number of candidates for vertex u . Condition 6.3.1 further limits the search. Condition 6.3.2 is pre-computed. Condition 6.3.3 searches the maximal vertices of i (a bounded number of vertices) for one whose bit position is smaller than that of u ($O(1)$) and is not a predecessor of u (typically available from the definition of P). Step 6.4 is $O(1)$ if the largest m satisfying 6.3.3 is saved for each iteration of the search. Step 6.5 is linear in the number of vertices.

Step 6.6 can be done in polynomial time. As each vertex u that is moved into j is maximal in j , any predecessors of u that were previously maximal in j are no longer maximal in j , and a successor v of u becomes minimal in $P \setminus j$ iff all of the predecessors of v are in j .

The loop in steps 6.7 through 6.7.4 will exit early if there are less than n_p 1's that can be moved out of j . This is a retry case in which the main loop exits without finding the next ideal; however, that can only happen a bounded number of times between any two ideals because the vertex candidate for u increases each retry and is bounded above by $2^{|P|}$. There is $O(1)$ setup processing for the next retry in steps 6.7.1.1 and 6.7.1.2. The largest bit position that can be moved out of j can be found in polynomial time by starting

from the largest maximal b in 6.7.2, moving b out of j in 6.7.3, and adjusting the boundary vertex sets in 6.7.4.

That leaves loop 6.8 through 6.8.4. This loop repeatedly moves the largest maximal out of j , moves the smallest minimal of the complement into j , and adjusts the maximal and minimal boundary sets, as long as the maximal moved out of j has a higher bit position than the minimal moved into j . By Lemma 7.1 and Lemma 7.2, the sequence of maximal vertices moved out of j decreases and the sequence of minimal vertices moved into j increases, so this loop must terminate. Step 6.9 sets the next ideal in $O(1)$. This completes the proof.

Theorem 7.7: The ideal exchange algorithm is correct. That is, the ideal exchange algorithm generates all ideals and only ideals of rank r in lex order and terminates in a finite number of steps.

Proof: By Lemma 7.6, the ideal exchange algorithm terminates in a finite number of steps. By Lemma 7.5, the ideal exchange algorithm generates only ideals of rank r in lex order. It remains to show that the ideal exchange algorithm generates every ideal of rank r . This will be proved by induction on the number of ideals generated. By Corollary 6.2, the first ideal generated in step 5 is the smallest ideal of rank r .

Let i_n be the most recently generated ideal, and let i_{n+1} be the next ideal of rank r in lex order. Since i_{n+1} is larger than i_n in lex order, the highest-order bit in which i_n and i_{n+1} differ must be a 0 in i_n and a 1 in i_{n+1} . Furthermore, this bit must be as low as possible (i.e., that allows an ideal of rank r to be made from i_n by changing only this bit and lower order bits). Step 6.3.3 continually tries the lowest candidate bit u until a value

is found for u that allows i_n to be extended to an ideal of rank r using only lower order bits. More precisely, step 6.5 moves u and all predecessors (a total of n_p vertices) not already in i_n to j , a “shadow” copy of i_n . By Lemma 7.4, this is the smallest number of 0 bits that must be changed to 1 so that the resulting characteristic vector is an ideal, and they must occur in the positions of u and all predecessors of u not already in i_n .

Then steps 6.7 and sub-steps attempts to move an equal number n_p of vertices at positions below u out of i_n . There are two cases:

Case 1: It is not possible to move n_p vertices from positions smaller than u out of i_n to form a new ideal of rank r . Since the portion of the algorithm to find the n_p vertices (if they exist) is greedy (by Corollary 7.3), this case is determined when there is no maximal vertex at a position below u to move out of i_n . Then the assignment $u_m = u + 1$ at step 6.7.1.1 causes the next value of u to be searched starting from one bit position higher than the value that was just tried. The retry count c_r , which is initialized to 0 in step 5, is incremented and the algorithm returns to the beginning of the main loop.

Case 2: It is possible to move n_p vertices from positions smaller than u out of i_n to form a new ideal of rank r . Let u_b be the value of u from step 6.5. By Lemma 7.2 (c) and Corollary 7.3, the largest bit pattern in lex order of positions below u_b moved out of i_n is found by successively moving the largest maximal vertex at a position below u_b out of i_n and adjusting the boundary vertex sets. Steps 6.7.2 through 6.7.4 do this n_p times. Then steps 6.7.2 through 6.7.4 find the highest bit pattern of n_p 1’s converted to 0’s over all possible ideals lexicographically greater than i_n . That is, the result is as small as possible among ideals that swap n_p vertices into and out of i_n at bit positions u and lower.

If step 6.7 and sub-steps are successful, the result is the smallest ideal j of rank r that is larger than i_n and only moves u and all of its predecessors into j . If $j > i_{n+1}$, the highest bit position B at which they differ must be below u_b and must be a 0 in i_{n+1} and a 1 in j . Since i_{n+1} is an ideal, b must be maximal in j . Let U be the set of vertex bit positions that are 0 in j and 1 in i_{n+1} , and let B be the set of vertex bit positions that are 1 in j and 0 in i_{n+1} . Then the largest bit position in U is smaller than the smallest bit position in B , the largest bit position in B is below u_b , $|U| = |B|$ is as large as possible, and the bit positions of U and B are respectively as small and as large in lex order as possible. By Lemma 7.2 and Corollary 7.3, the vertices in U and B are found by successively moving the largest maximal vertex b of j below u_b out of j , moving the smallest vertex u not already in j into j , and re-computing the boundary, as long as $u < b$. Steps 6.8 and sub-steps accomplish this; the effect is to move the 1's lower and the 0's higher. By Lemma 7.1, the sequence of 0 bits to swap in increases and the sequence of 1 bits to swap out decreases. By Lemma 7.2 and Corollary 7.3, the result is the best possible ideal; i.e., at the end of the process, $j = i_{n+1}$ is the smallest ideal in lex order larger than i_n . The algorithm moves j to i , saves this new ideal in the array I of ideals, and increments the Whitney number $W[r]$, which is initialized to 1 for the initial ideal in step 5. The bit position u_m used to start the search for the next u is set to 0, its default value. This extends the induction.

The algorithm halts when the largest ideal in lex order, i_{max} , has been generated. Since every ideal is generated in lex order, i_{max} is guaranteed to be found as the last ideal generated. This completes the proof.

It should be noted that the max ideal generation algorithm used to generate i_{max} assumes that P is a Boolean lattice. The ideal exchange algorithm assumes that i_{max} is input. Also note that steps 4 and 6.3.2 are specific to a Boolean lattice but are only present for efficiency reasons. These steps can be eliminated. The resulting algorithm will work for any ranked poset P but may have more retries than the algorithm as written for $P = B_k$.

CHAPTER 8: NORMALITY TEST AND VERIFICATION PROGRAMS

Two computer programs were developed to test the normality of $\mathcal{J}(B_k)$. The first program, “normaltest”, performs the ideal equivalence class construction, sets up the bipartite networks, runs the MPM flow algorithm and outputs the results. The second program, “checker”, reads in the data generated by normaltest and performs checks to verify the consistency of the input data and to test $\mathcal{J}(B_k)$ for normality.

The posets tested for normality were $\mathcal{J}(B_k)$ for $k \in \{1, \dots, 6\}$. Case 6 was by far the most computationally challenging case. The challenges posed included large integers to store each ideal, large data sets to store the set of all ideal equivalence classes, and long computation times to find the equivalence classes and perform the flow calculations. These problems were addressed in part by the structure of the computer programs and in part by the development environment. We discuss each of these in turn.

The development environment used was the computer system of the University of California, Riverside, Department of Computer Science and Engineering. The computer was an SGI ALTIX 4700, consisting of 32 Itanium 1.594 GHz processors and 64 GB RAM, using the Linux command line operating system. The Intel icc C compiler was used to generate the executable program from the source, using the following command:

```
/opt/intel/cc/9.1.042/bin/icc normaltest.cpp
```

This takes the C++ source program, normaltest.cpp, and generates the executable file a.out, which is then renamed normaltest. A similar process was used to compile the

C++ program checker.cpp via the command `/opt/intel/cc/9.1.042/bin/icc checker.cpp` and renaming the a.out file produced as checker.

The source files normaltest.cpp and checker.cpp are listed in Appendices B and D, respectively. Both appendices are available as separate electronic files. The remainder of this chapter discusses the two programs.

The program normaltest is run entering the command `./normaltest` followed by a carriage return. The normaltest program begins by requesting input data. The first input prompt is

enter E for exchange algorithm or S for selective algorithm

The user then enters E (or e) to run the exchange algorithm, or S (or s) to run the selective algorithm. The next input prompt is

make classes? (Y or N)

The user enters Y (or y) to make equivalence classes from the generated ideals, or N (or n) to run the program on the individual ideals. The next input prompt is

input n:

The user then enters an integer value in the range $\{1, \dots, 7\}$. The program then computes the maximum rank 2^n and outputs the following prompts:

input integer ranks such that $0 \leq \text{lowest rank} < \text{highest rank} \leq 2^n$

lowest rank=

highest rank=

After the user enter the desired ranks, the last input prompt is displayed:

trace on? (Y or N)

The user then enters Y (or y) to cause the generation of a trace file, or N (or n) to decline this option. Trace files give detailed information about the computations performed by the MPM flow algorithm and are intended only if a detailed analysis is desired. For $n = 6$, the trace files are on the order of a million lines of text, so one should use the “trace on” option judiciously.

As an example, the following prompts and inputs (shown in **bold**) set up the normaltest program to use the selective ideal generation algorithm to build equivalence classes of ideals and run the MPM flow algorithm for all ranks of $\mathcal{J}(B_5)$ with no trace file:

```

enter E for exchange algorithm or S for selective algorithm S
make classes? (Y or N) Y
input n: 5
input integer ranks such that  $0 \leq \text{lowest rank} < \text{highest rank} \leq 32$ 
lowest rank= 0
highest rank= 32
trace on? (Y or N) N

```

Note that the selective ideal generation algorithm run over all ranks is just the traditional ideal generation algorithm.

The normaltest program then runs without any further input from the user. For $\mathcal{J}(B_1)$ through $\mathcal{J}(B_5)$, the program terminates in real time even when all ranks are specified. For one pair of adjacent ranks near the middle of $\mathcal{J}(B_6)$, however, the program requires days of execution time. By Proposition 4.1, $|\mathcal{J}(B_k)|$ grows nearly as fast as 2^{2^k} , and this growth becomes apparent as k goes from 5 to 6. This dramatic increase in execution time had a profound effect on the debugging and execution of the normaltest program for the case $\mathcal{J}(B_6)$. It may be qualitatively stated that, for this problem, “5 is easy and 6 is hard”. See Chapter 9 for execution times of the cases run in this study.

Because the execution times can be large, the normaltest program provides outputs to the terminal screen during execution to give the user a sense of the progress of the program, in addition to the output files generated.

The outputs produced by the normaltest program will be discussed next. There are three output destinations. As noted above, data are output to the terminal screen. The output file output.txt is generated along with the file data.txt that is input to the checker program.

Upon receipt of the input discussed above, the normaltest program begins generating ideals (and, if so specified, equivalence classes of ideals). The normaltest program outputs a line to the terminal screen whenever a new ideal equivalence class is generated. New equivalence classes are created much more quickly at the beginning of this process than at the end, but the user can at least watch the progress of this part of the program. The user is informed that all ideals have been generated when the message “done generating ideals” is displayed on the terminal screen.

The program then sets up the bipartite flows and executes the MPM algorithm. This is another computationally intensive process that involves phases as discussed in Chapter 5, so the program displays a message when processing for a phase begins and again when it ends.

At the end of the normaltest program, the results are displayed on the terminal screen. These include whether or not $\mathcal{J}(B_k)$ is normal over the specified ranks, the calculated Whitney numbers, the Dedekind number $|\mathcal{J}(B_k)|$, and whether or not $\mathcal{J}(B_k)$ is

unimodal. The last two outputs are only generated if either all the ranks from the lower half of $\mathcal{J}(B_k)$ or all the ranks from the upper half of $\mathcal{J}(B_k)$ were run.

The file output.txt is intended to be a human-readable version of the results. Identifying information is displayed based on the inputs for n and the ranks of interest. This is followed by the time to generate the ideal equivalence classes and (if the ideal exchange algorithm was used) the number of retries (see Chapter 7) per rank. These data are used to analyze the performance of the program options as discussed in Chapter 9. Then, for each pair of adjacent ranks, the flow calculation time is given, followed by a list of the edges with capacities and flows. At the end of the output.txt file, the results are listed. These results include whether $\mathcal{J}(B_k)$ is normal over ranks specified, the Whitney numbers of the specified ranks, and if sufficient ranks were specified, the Dedekind number $|\mathcal{J}(B_k)|$ and whether $\mathcal{J}(B_k)$ is unimodal. The specific output.txt files generated for each case run in this study are shown in Appendix C, available as a separate electronic file.

The file data.txt is a more simply structured version of the results that is input to the checker program. The format of this file will be discussed later in this chapter.

These files are generated as follows. First, the ideals are generated. As they are generated, new equivalence classes, and edges between equivalence classes, are created as necessary. The weight of each equivalence class is its cardinality. Then, for each pair of adjacent ranks, the normalized bipartite flow problem is created, scaled up to integer capacities, and solved using the MPM algorithm as discussed in Chapter 5.

Before discussing the checker program, the resolution of some computational challenges pertinent to the normaltest program will be discussed. To handle large data sets, the characteristic vectors were represented as bit strings. This saves considerable storage at the expense of the need for long words. The long long data type, which is a 64-bit unsigned integer, was used extensively for this reason. In addition, a composite data type was created that spreads the characteristic vector bits over multiple words. This necessitated the need to perform multi-word arithmetic. Fortunately, the need for this was limited to the setting and testing of bits; no arithmetic operations such as addition or multiplication were necessary on the characteristic vectors.

A more subtle problem arose in the implementation of the MPM algorithm. The statement of the algorithm in [Even 1979] did not address some necessary bookkeeping details. The discovery and implementation of these required details was all the more challenging given the need to run long data sets over many days and wade through millions of lines of trace files to find a new error and fix it. The bookkeeping features that were discovered and added in this manner occur when a vertex of 0 potential is eliminated from the active vertex set. It was found to be necessary to prune the layered network of all successors and predecessors, propagating this out to t and back to s for any other vertices that no longer are on any path from s to t . It was also discovered that the input and output potential of remaining vertices needed to be adjusted for the deletion of each vertex removed from the active vertex set, either because it has 0 potential or because it is no longer on a path from s to t .

One implementation detail that was easy to solve was the execution of versions of the program in parallel. This gives the improved calendar time for solution of many flow problems and was the motivation behind the development of the new selective ideal generation and ideal exchange algorithms. To implement parallel execution, the normaltest program was simply copied into multiple subdirectories, one for each instance of the program to be executed in parallel.

Once the data files have been generated by the normaltest program, the program checker is run using the command `./checker` followed by a carriage return. There is no user input; program checker searches for a file named `data.txt` in the same directory. This file is one of the files generated as output by the normaltest program, and it provides all the input data needed by the checker program. In the parlance of computability theory, the `data.txt` file provides the certificate that demonstrates a solution to the bipartite normalized flow problem.

The structure of the file `data.txt` will be described next. The specific `data.txt` files, output from the normaltest program and input to the checker program, are listed in Appendix E, which is available as a separate electronic file.

The first entry in `data.txt` is the value of n input to the normaltest program. Then, for each pair of adjacent ranks, the lower rank and Whitney number are input, followed by the higher rank and Whitney number. Next comes the number of vertices (equivalence classes) in the lower rank. For each vertex, the vertex number, weight and number of edges coming into the vertex are read in. The vertex numbers are sequential, and a consistency check is performed to ensure that this is the case in the `data.txt` file.

For each incoming edge, the edge number and the flow on the edge reported by the normaltest program are read in. The total flow into the edge is calculated and compared to the total flow into the edge read in from the data.txt file. This structure is repeated for the outgoing edges. A check is performed to ensure that flow is conserved for each vertex; i.e., the total flow in equals the total flow out.

Once the data for all vertices in the lower rank have been read in, the same data are read in and processed in a similar manner for all vertices in the upper rank. An additional check is made to verify that the flow specified on an edge coming into a vertex in the upper rank is the same as the flow specified on the same edge going out of a vertex in the lower rank. As for vertices in the lower rank, each vertex in the upper rank is checked to ensure that flow is conserved.

Then the data for the edges going out of s , and the data for the edges coming into t , are read in and processed in a similar manner. Once again, the flow on an edge going out of s or coming into t is compared to the flow read in for the same edge when input flows into vertices in the lower rank, or output flows from vertices in the upper rank, respectively, were processed.

The final input is the calculated normal flow reported by the normaltest program. This value must match the product of the two Whitney numbers read in, the total flow out of s , the total flow into all vertices of the lower rank, the total flow out of all vertices of the lower rank, the total flow into all vertices of the upper rank, the total flow out of all vertices of the upper rank, and the total flow into t . These values must all be equal in

order for the checker program to verify that the flow calculated by the normaltest program was indeed a normalized flow.

The output of the checker program is a data file named checkfile.txt. The format of this file is as follows. Identifying information (n and the ranks of interest) is listed first. Then the main result is listed. For successful runs, the message displayed is “Flow is normal and calculations are consistent”. A “Flow calculation mismatch” message indicates failure. The rest of the checkfile.txt file contains supporting data. All calculated total flow values (normal flow, flow out of s , flow into the lower rank, flow out of the lower rank, flow into the upper rank, flow out of the upper rank, flow into t) are displayed. Then, if the Whitney numbers read in match the values calculated from the sum of the vertex weights, the message “Whitney numbers check for both ranks” is displayed. A “Whitney number mismatch” indicates an error. Then the message “Flow is conserved for all vertices” is displayed if true. The message “Flow is not conserved for all vertices” indicates an error. The checkfile.txt file ends with the flow into, and out of, each vertex as supporting data.

Appendix F is a separate electronic file that contains the checkfile.txt files for each case run in this study. All results considered in this study were successful, indicating that $\mathcal{J}(B_k)$ is normal for integer values of k from 1 through 6.

CHAPTER 9: RESULTS

This chapter contains the main empirical results from the normaltest and checker computer programs. These programs show that $\mathcal{J}(B_k)$ is normal and unimodal, and therefore strongly Sperner, for integer values of k from 1 through 6.

The strategy used to verify the results, especially for the computationally intensive case $k = 6$, is important enough to be discussed prior to the presentation of the results. For $k = 1$ through 4, the results can be calculated by hand, and are done so in this chapter. This provides a baseline against which the computer solutions can be compared. For $k = 5$ or 6, the computer runs become the baseline, with spot checks performed to ensure consistency. It is not feasible to perform hand calculations over all ranks for $k = 5$ or $k = 6$, but hand calculations are still possible for ranks near the bottom and top.

The above strategy focuses on extension of normality tests to higher-order cases. An additional strategy was used to establish the correctness of the new ideal exchange algorithm. The first step was to prove the correctness of the new algorithm. This was done in Chapter 7. Then the normaltest program was run over all the data sets using both the selective ideal generation algorithm and the ideal exchange algorithm. Spot checks were performed on details such as the number of vertices and edges in the quotient network and the total flow, in addition to the main checks (correctness of Dedekind numbers and flows).

This strategy confirmed that the results were consistent and correct. The strategy combined theory with empirical evidence. As L. H. Harper points out, mathematics and computer science are empirical sciences as much as they are formal theories.

The rest of this chapter provides the detailed analyses of the results. First we will show a non-Sperner example and verify that the flow algorithm does not produce a normalized flow. Figure 9.1 shows a graded poset that is not Sperner, and therefore not normal, because the largest antichain $\{a, b, e, f\}$ does not lie on one rank.

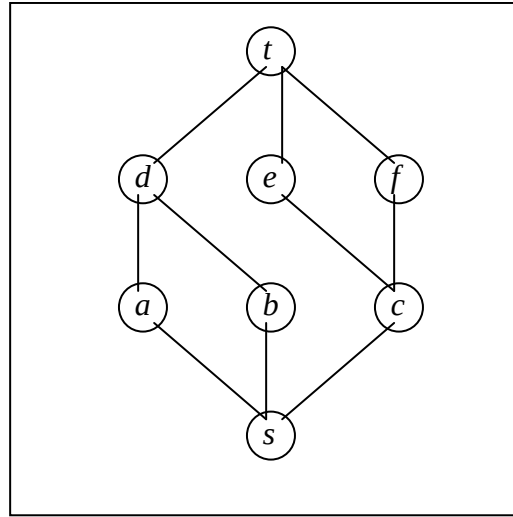


Figure 9.1. A Graded Poset that is Not Sperner

If we consider the bipartite flow problem on the middle two ranks with added vertices s and t , we have a diagram identical to the entire poset. For a unit flow to exist, there must be a flow of $1/3$ out of each of a , b , and c and a flow of $1/3$ into each of d , e , and f . But then each of a and b must give all of its flow to d , and so d would have a flow of $2/3$ coming into it, a contradiction.

Hand solutions for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 4$, will be given next, starting with the ideals given in Table 9.1 as characteristic vectors. The next proposition shows that the elements of $\mathcal{J}(B_k)$ in lex order include the elements of $\mathcal{J}(B_1)$, $\dots$, $\mathcal{J}(B_{k-1})$ as initial segments.

Proposition 9.1: The lex ordering of $\mathcal{J}(B_k)$ has nested solutions; i.e., for $k > 0$, $\mathcal{J}(B_{k-1})$ occurs in lex order (with leading 0's) in the first $|\mathcal{J}(B_{k-1})|$ positions of $\mathcal{J}(B_k)$.

Proof: Each ideal i in $\mathcal{J}(B_{k-1})$ “is” an ideal of $\mathcal{J}(B_k)$. (That is, the inclusion map injects i into $\mathcal{J}(B_k)$ using only the $k - 1$ smallest atoms $2^0, 2^1, \dots, 2^{k-2}$.) Conversely, every ideal of $\mathcal{J}(B_k)$ that does not use the largest atom 2^{k-1} “is” an ideal of $\mathcal{J}(B_{k-1})$. Thus the ideals of $\mathcal{J}(B_k)$ that do not use the largest atom 2^{k-1} are precisely the ideals of $\mathcal{J}(B_{k-1})$. Since the other ideals of $\mathcal{J}(B_k)$ use the largest atom 2^{k-1} , the ideals of $\mathcal{J}(B_{k-1})$ occur in lex order as the initial segment of $\mathcal{J}(B_k)$.

Thus the ideals of $\mathcal{J}(B_4)$ include the ideals of $\mathcal{J}(B_1)$ as an initial segment, the ideals of $\mathcal{J}(B_2)$ as a larger initial segment, and the ideals of $\mathcal{J}(B_3)$ as a still larger initial segment. This is reflected in Table 9.1 below. The entry under n (row number) gives the numerical order of the ideal in lex order, so that $f: \mathcal{J}(B_k) \rightarrow \mathcal{N}_{|\mathcal{J}(B_k)|}$ with $f^{-1}(n) = i_n$, the n^{th} ideal of $\mathcal{J}(B_k)$, is a total extension of the inclusion ordering of $\mathcal{J}(B_k)$. The entry under k is the smallest value of k such that $\mathcal{J}(B_k)$ contains the ideal (see Proposition 9.1). The table shows a border where the value of k increases, giving the boundaries for $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$ and $\mathcal{J}(B_3)$. The entry under rank gives the number of 1 bits (cardinality) of the n^{th} ideal of $\mathcal{J}(B_k)$, which is shown next in binary followed by the corresponding decimal value under “value”. The immediate predecessors and successors for the ideal are shown as their numerical order (row number) under n . The entry under “dual” is the numerical order (row number) entry of the anti-automorphic ideal defined by reversal and complementation in Definition 4.11.

Table 9.1. Ideals of $\mathcal{I}(B_1)$, $\mathcal{I}(B_2)$, $\mathcal{I}(B_3)$, and $\mathcal{I}(B_4)$ in Lex Order (1 of 6)

n	k	rank	n^{th} ideal of $\mathcal{I}(B_k)$	value	predecessors	successors	dual
0	0	0	0000000000000000	0		1	167
1	0	1	0000000000000001	1	0	2, 3, 6, 20	166
2	1	2	0000000000000011	3	1	4, 7, 21	147
3	2	2	0000000000000101	5	1	4, 8, 22	161
4	2	3	0000000000000111	7	2, 3	5, 9, 23	133
5	2	4	0000000000001111	15	4	10, 24	83
6	3	2	0000000000010001	17	1	7, 8, 25	164
7	3	3	0000000000010011	19	2, 6	9, 11, 26	144
8	3	3	0000000000010101	21	3, 6	9, 14, 27	158
9	3	4	0000000000010111	23	4, 7, 8	10, 12, 15, 28	128
10	3	5	0000000000011111	31	5, 9	13, 16, 29	77
11	3	4	0000000000110011	51	7	12, 30	139
12	3	5	0000000000110111	55	9, 11	13, 17, 31	108
13	3	6	0000000000111111	63	10, 12	18, 32	52
14	3	4	0000000001010101	85	8	15, 33	153
15	3	5	0000000001010111	87	9, 14	16, 17, 34	119
16	3	6	0000000001011111	95	10, 15	18, 35	66
17	3	6	0000000001110111	119	12, 15	18, 36	97
18	3	7	0000000001111111	127	13, 16, 17	19, 37	38
19	3	8	0000000011111111	255	18	38	19
20	4	2	0000000100000001	257	1	21, 22, 25	165
21	4	3	0000000100000011	259	2, 20	23, 26, 39	146
22	4	3	0000000100000101	261	3, 20	23, 27, 53	160
23	4	4	0000000100000111	263	4, 21, 22	24, 28, 40, 54	132
24	4	5	0000000100001111	271	5, 23	29, 41, 55	82
25	4	3	0000000100010001	273	6, 20	26, 27, 84	163
26	4	4	0000000100010011	275	7, 21, 25	28, 30, 42, 85	143
27	4	4	0000000100010101	277	8, 22, 25	28, 33, 56, 86	157
28	4	5	0000000100010111	279	9, 23, 26, 27	29, 31, 34, 43, 57, 87	127
29	4	6	0000000100011111	287	10, 24, 28	32, 35, 44, 58, 88	76
30	4	5	0000000100110011	307	11, 26	31, 45, 89	138
31	4	6	0000000100110111	311	12, 28, 30	32, 36, 46, 59, 90	107
32	4	7	0000000100111111	319	13, 29, 31	37, 47, 60, 91	51
33	4	5	0000000101010101	341	14, 27	34, 61, 92	152
34	4	6	0000000101010111	343	15, 28, 33	35, 36, 48, 62, 93	118

Table 9.1. Ideals of $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$, $\mathcal{J}(B_3)$, and $\mathcal{J}(B_4)$ in Lex Order (2 of 6)

n	k	rank	n^{th} ideal of $\mathcal{J}(B_k)$	value	predecessors	successors	dual
35	4	7	0000000101011111	351	16, 29, 34	37, 49, 63, 94	65
36	4	7	0000000101110111	375	17, 31, 34	37, 50, 64, 95	96
37	4	8	0000000101111111	383	18, 32, 35, 36	38, 51, 65, 96	37
38	4	9	0000000111111111	511	19, 37	52, 66, 97	18
39	4	4	0000001100000011	771	21	40, 42	145
40	4	5	0000001100000111	775	23, 39	41, 43, 67	130
41	4	6	0000001100001111	783	24, 40	44, 68	80
42	4	5	0000001100010011	787	26, 39	43, 45, 98	141
43	4	6	0000001100010111	791	28, 40, 42	44, 46, 48, 69, 99	123
44	4	7	0000001100011111	799	29, 41, 43	47, 49, 70, 100	72
45	4	6	0000001100110011	819	30, 42	46, 101	136
46	4	7	0000001100110111	823	31, 43, 45	47, 50, 71, 102	103
47	4	8	0000001100111111	831	32, 44, 46	51, 72, 103	47
48	4	7	0000001101010111	855	34, 43	49, 50, 73, 104	113
49	4	8	0000001101011111	863	35, 44, 48	51, 74, 105	60
50	4	8	0000001101110111	887	36, 46, 48	51, 75, 106	91
51	4	9	0000001101111111	895	37, 47, 49, 50	52, 76, 107	32
52	4	10	0000001111111111	1023	38, 51	77, 108	13
53	4	4	0000010100000101	1285	22	54, 56	159
54	4	5	0000010100000111	1287	23, 53	55, 57, 67	131
55	4	6	0000010100001111	1295	24, 54	58, 68	81
56	4	5	0000010100010101	1301	27, 53	57, 61, 109	155
57	4	6	0000010100010111	1303	28, 54, 56	58, 59, 62, 69, 110	125
58	4	7	0000010100011111	1311	29, 55, 57	60, 63, 70, 111	74
59	4	7	0000010100110111	1335	31, 57	60, 64, 71, 112	105
60	4	8	0000010100111111	1343	32, 58, 59	65, 72, 113	49
61	4	6	0000010101010101	1365	33, 56	62, 114	150
62	4	7	0000010101010111	1367	34, 57, 61	63, 64, 73, 115	116
63	4	8	0000010101011111	1375	35, 58, 62	65, 74, 116	63
64	4	8	0000010101110111	1399	36, 59, 62	65, 75, 117	94
65	4	9	0000010101111111	1407	37, 60, 63, 64	66, 76, 118	35
66	4	10	0000010111111111	1535	38, 65	77, 119	16
67	4	6	0000011100000111	1799	40, 54	68, 69	129
68	4	7	0000011100001111	1807	41, 55, 67	70, 78	79
69	4	7	0000011100010111	1815	43, 57, 67	70, 71, 73, 120	121
70	4	8	0000011100011111	1823	44, 58, 68, 69	72, 74, 79, 121	70
71	4	8	0000011100110111	1847	46, 59, 69	72, 75, 122	100

Table 9.1. Ideals of $\mathcal{I}(B_1)$, $\mathcal{I}(B_2)$, $\mathcal{I}(B_3)$, and $\mathcal{I}(B_4)$ in Lex Order (3 of 6)

n	k	rank	n^{th} ideal of $\mathcal{I}(B_k)$	value	predecessors	successors	dual
72	4	9	0000011100111111	1855	47, 60, 70, 71	76, 80, 123	44
73	4	8	0000011101010111	1879	48, 62, 69	74, 75, 124	111
74	4	9	0000011101011111	1887	49, 63, 70, 73	76, 81, 125	58
75	4	9	0000011101110111	1911	50, 64, 71, 73	76, 126	88
76	4	10	0000011101111111	1919	51, 65, 72, 74, 75	77, 82, 127	29
77	4	11	0000011111111111	2047	52, 66, 76	83, 128	10
78	4	8	0000111100001111	3855	68	79	78
79	4	7	0000111100011111	3871	70, 78	80, 81, 129	68
80	4	10	0000111100111111	3903	72, 79	82, 130	41
81	4	10	0000111101011111	3935	74, 79	82, 131	55
82	4	11	0000111101111111	3967	76, 80, 81	83, 132	24
83	4	12	0000111111111111	4095	77, 82	133	5
84	4	4	0001000100010001	4369	25	85, 86	162
85	4	5	0001000100010011	4371	26, 84	87, 89, 98	142
86	4	5	0001000100010101	4373	27, 84	87, 92, 109	156
87	4	6	0001000100010111	4375	28, 85, 86	88, 90, 93, 99, 110	126
88	4	7	0001000100011111	4383	29, 87	91, 94, 100, 111	75
89	4	6	0001000100110011	4403	30, 85	90, 101	137
90	4	7	0001000100110111	4407	31, 87, 89	91, 95, 102, 112	106
91	4	8	0001000100111111	4415	32, 88, 90	96, 103, 113	50
92	4	6	0001000101010101	4437	33, 86	93, 114	151
93	4	7	0001000101010111	4439	34, 87, 92	94, 95, 104, 115	117
94	4	8	0001000101011111	4447	35, 88, 93	96, 105, 116	64
95	4	8	0001000101110111	4471	36, 90, 93	96, 106, 117	95
96	4	9	0001000101111111	4479	37, 91, 94, 95	97, 107, 118	36
97	4	10	0001000111111111	4607	38, 96	108, 119	17
98	4	6	0001001100010011	4883	42, 85	99, 101	140
99	4	7	0001001100010111	4887	43, 87, 98	100, 102, 104, 120	122
100	4	8	0001001100011111	4895	44, 88, 99	103, 105, 121	71
101	4	7	0001001100110011	4915	45, 89, 98	102, 134	135
102	4	8	0001001100110111	4919	46, 90, 99, 101	103, 106, 122, 135	102
103	4	9	0001001100111111	4927	47, 91, 100, 102	107, 123, 136	46
104	4	8	0001001101010111	4951	48, 93, 99	105, 106, 124	112

Table 9.1. Ideals of $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$, $\mathcal{J}(B_3)$, and $\mathcal{J}(B_4)$ in Lex Order (4 of 6)

n	k	rank	n^{th} ideal of $\mathcal{J}(B_k)$	value	predecessors	successors	dual
105	4	9	0001001101011111	4959	49, 94, 100, 104	107, 125	59
106	4	9	0001001101110111	4983	50, 95, 102, 104	107, 126, 137	90
107	4	10	0001001101111111	4991	51, 96, 103, 105, 106	108, 127, 138	31
108	4	11	0001001111111111	5119	52, 97, 107	128, 139	12
109	4	6	0001010100010101	5397	56, 86	110, 114	154
110	4	7	0001010100010111	5399	57, 87, 109	111, 112, 115, 120	124
111	4	8	0001010100011111	5407	58, 88, 110	113, 116, 121	73
112	4	8	0001010100110111	5431	59, 90, 110	113, 117, 122	104
113	4	9	0001010100111111	5439	60, 91, 111, 112	118, 123	48
114	4	7	0001010101010101	5461	61, 92, 109	115, 148	149
115	4	8	0001010101010111	5463	62, 93, 110, 114	116, 117, 124, 149	115
116	4	9	0001010101011111	5471	63, 94, 111, 115	118, 125, 150	62
117	4	9	0001010101110111	5495	64, 95, 112, 115	118, 126, 151	93
118	4	10	0001010101111111	5503	65, 96, 113, 116, 117	119, 127, 152	34
119	4	11	0001010111111111	5631	66, 97, 118	128, 153	15
120	4	8	0001011100010111	5911	69, 99, 110	121, 122, 124	120
121	4	9	0001011100011111	5919	70, 100, 111, 120	123, 125, 129	69
122	4	9	0001011100110111	5943	71, 102, 112, 120	123, 126, 140	99
123	4	10	0001011100111111	5951	72, 103, 113, 121, 122	127, 130, 141	43
124	4	9	0001011101010111	5975	73, 104, 115, 120	125, 126, 154	110
125	4	10	0001011101011111	5983	74, 105, 116, 121, 124	127, 131, 155	57
126	4	10	0001011101110111	6007	75, 106, 117, 122, 124	127, 142, 156	87
127	4	11	0001011101111111	6015	76, 107, 118, 123, 125, 126	128, 132, 143, 157	28

Table 9.1. Ideals of $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$, $\mathcal{J}(B_3)$, and $\mathcal{J}(B_4)$ in Lex Order (5 of 6)

n	k	rank	n^{th} ideal of $\mathcal{J}(B_k)$	value	predecessors	successors	dual
128	4	12	0001011111111111	6143	77, 108, 119, 127	133, 144, 158	9
129	4	10	0001111100011111	7967	79, 121	130, 131	67
130	4	11	0001111100111111	7999	80, 123, 129	132, 145	40
131	4	11	0001111101011111	8031	81, 125, 129	132, 159	54
132	4	12	0001111101111111	8063	82, 127, 130, 131	133, 146, 160	23
133	4	13	0001111111111111	8191	83, 128, 132	147, 161	4
134	4	8	0011001100110011	13107	101	135	134
135	4	9	0011001100110111	13111	102, 134	136, 137, 140	101
136	4	10	0011001100111111	13119	103, 135	138, 141	45
137	4	10	0011001101110111	13175	106, 135	138, 142	89
138	4	11	0011001101111111	13183	107, 136, 137	139, 143	30
139	4	12	0011001111111111	13311	108, 138	144	11
140	4	10	0011011100110111	14135	122, 135	141, 142	98
141	4	11	0011011100111111	14143	123, 136, 140	143, 145	42
142	4	11	0011011101110111	14199	126, 137, 140	143, 162	85
143	4	12	0011011101111111	14207	127, 138, 141, 142	144, 146, 163	26
144	4	13	0011011111111111	14335	128, 139, 143	147, 164	7
145	4	12	0011111100111111	16191	130, 141	146	39
146	4	13	0011111101111111	16255	132, 143, 145	147, 165	21
147	4	14	0011111111111111	16383	133, 144, 146	166	2
148	4	8	0101010101010101	21845	114	149	148
149	4	9	0101010101010111	21847	115, 148	150, 151, 154	114
150	4	10	0101010101011111	21855	116, 149	152, 155	61
151	4	10	0101010101110111	21879	117, 149	152, 156	92
152	4	11	0101010101111111	21887	118, 150, 151	153, 157	33
153	4	12	0101010111111111	22015	119, 152	158	14
154	4	10	0101011101010111	22359	124, 149	155, 156	109
155	4	11	0101011101011111	22367	125, 150, 154	157, 159	56
156	4	11	0101011101110111	22391	126, 151, 154	157, 162	86
157	4	12	0101011101111111	22399	127, 152, 155, 156	158, 160, 163	27
158	4	13	0101011111111111	22527	128, 153, 157	161, 164	8
159	4	12	0101111101011111	24415	131, 155	160	53
160	4	13	0101111101111111	24447	132, 157, 159	161, 165	22
161	4	14	0101111111111111	24575	133, 158, 160	166	3
162	4	12	0111011101110111	30583	142, 156	163	84

Table 9.1. Ideals of $\mathcal{J}(B_1)$, $\mathcal{J}(B_2)$, $\mathcal{J}(B_3)$, and $\mathcal{J}(B_4)$ in Lex Order (6 of 6)

n	k	rank	n^{th} ideal of $\mathcal{J}(B_k)$	value	predecessors	successors	dual
163	4	13	0111011101111111	30591	143, 157, 162	164, 165	25
164	4	14	0111011111111111	30719	144, 158, 163	166	6
165	4	14	0111111101111111	32639	146, 160, 163	166	20
166	4	15	0111111111111111	32767	147, 161, 164, 165	167	1
167	4	16	1111111111111111	65535	166		0

In the following, ideals of $\mathcal{J}(B_4)$ will be referenced by their numeric order (row number in Table 9.1). Table 9.1 was generated by hand, using the traditional ideal generation algorithm. This is potentially an error-prone process. However, the computed values match the values in the table, reinforcing each other. Values such as the Dedekind numbers (see Appendix A) also agree with previously known values.

Definition 9.2: Let i be an ideal in $\mathcal{J}(B_k)$. The *dual* of i is $\delta(i)$, where δ is the anti-automorphism operator $\delta(i) = (i^R)'$ from Definition 4.11. If $i = \delta(i)$, i is *self-dual*. Let $c = \{v_1 < \dots < v_{l(c)}\}$ be a chain in $\mathcal{J}(B_k)$. The *dual chain* $d = \{w_1 < \dots < w_{l(d)}\}$ of c satisfies $l(c) = l(d)$ and $w_{l(c)+1-j} = \delta(v_j)$. If $d = c$, c is *self-dual*.

Note that the dual of i in $\mathcal{J}(B_k)$ is not the same as the dual of i in $\mathcal{J}(B_n)$ for $n > k$.

Analysis of Table 9.1 shows that there are 12 ideals of rank 8 (the middle rank for $\mathcal{J}(B_4)$) that are not self-dual. These fall into the following six dual pairs:

- 49 and 60
- 50 and 91
- 64 and 94
- 71 and 100
- 73 and 111
- 104 and 112

Upon examining these ideals, we see that they are all isomorphic to each other. All have the shape shown in Figure 9.2. In fact, they form the entire equivalence class. The number inside the circle is the class representative (the smallest vertex in the class in lex order). The four vertices of rank 1 include two that have two successors each and two that have one successor each. There are $\binom{4}{2} = 6$ ways to partition these vertices this way. Given any such partition, a fixed vertex with one successor can be paired with either of the two vertices with two successors, so the size of this equivalence class is 12, as shown by the number outside the circle. This style of argument will be used in the hand examples to follow.

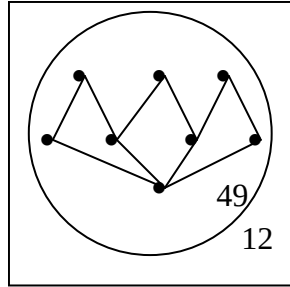


Figure 9.2. Equivalence Class of Non-Self-Dual Ideals of Rank 8 in $\mathcal{J}(B_4)$

If every entry in the middle rank of $\mathcal{J}(B_k)$ were self-dual, then any chain construction on either half of $\mathcal{J}(B_k)$ would automatically extend across the other half of $\mathcal{J}(B_k)$. If a symmetric chain decomposition exists for either half of $\mathcal{J}(B_k)$ in general, then $\mathcal{J}(B_k)$ would be Sperner. If, in addition, the entries in the middle rank were all self-dual in general, then $\mathcal{J}(B_k)$ would be strongly Sperner. The non-self-dual entries of rank 8 in $\mathcal{J}(B_4)$ show that this inference cannot be made.

This problem was solved for the analysis of normality via flowmorphisms to reduce the ideal to its quotient by the symmetric group. However, the question of an appropriate morphism for symmetric chain decompositions is open (see Chapter 10).

Hand solutions will be given next to show that $\mathcal{I}(B_0)$, $\mathcal{I}(B_1)$, $\mathcal{I}(B_2)$, $\mathcal{I}(B_3)$, and $\mathcal{I}(B_4)$ are normal. The max flow problem will be solved on $\mathcal{I}(B_k) / S_k$, producing a normal flow in each case. By Proposition 9.1, Table 9.1 can be used to build the solution for each case.

Figure 9.3 shows B_0 and $\mathcal{I}(B_0)$. B_0 has no atoms and consists solely of the bottom vertex. The ideals are the null ideal $\emptyset$ and B_0 . These two ideals are of different ranks, so $\mathcal{I}(B_0) / S_0 = \mathcal{I}(B_0)$. The normal flow is trivially 1, as shown by the unit flow on the edge connecting the two ideals. A symmetric chain decomposition of $\mathcal{I}(B_0)$ is also trivial. Note that, unlike $\mathcal{I}(B_k)$ or $\mathcal{I}(B_k) / S_k$ for $k > 0$, $\mathcal{I}(B_0) = \mathcal{I}(B_0) / S_0$ does not have a middle rank. Since this case is trivial, no computer solution was generated for it.

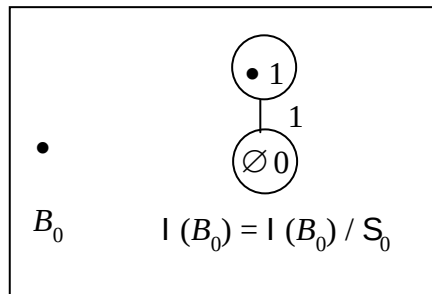


Figure 9.3. Normal Flow Solution for $\mathcal{I}(B_0)$

Figure 9.4 shows B_1 and $\mathcal{I}(B_1)$. B_1 consists of the bottom vertex and one atom, which is the top vertex. The ideals are the null ideal $\emptyset$, the singleton set consisting of the

bottom element, all all of B_1 . Since no rank of $\mathcal{J}(B_1)$ has more than one ideal, $\mathcal{J}(B_1) / S_1 = \mathcal{J}(B_1)$. There are two pairs of adjacent ranks, with a trivial normal flow of 1 in each case, as shown on the edge connecting ranks 0 and 1 and on the edge connecting ranks 1 and 2. A symmetric chain decomposition of $\mathcal{J}(B_1)$ is again trivial. A computer solution to the normal flow problem for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 6$, is given in Appendices C (normaltest output file), E (checker input file), and F (checker output file). These Appendices are very large, covering $\mathcal{J}(B_1)$ through $\mathcal{J}(B_6)$, and are available as separate electronic files.

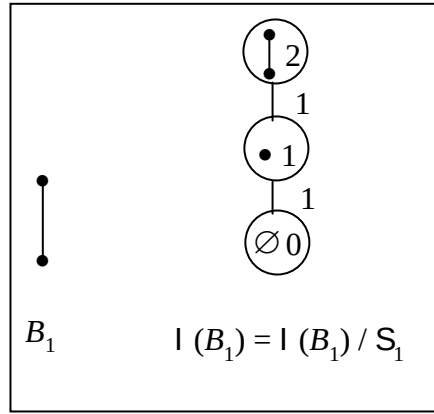


Figure 9.4. Normal Flow Solution for $\mathcal{J}(B_1)$

Figure 9.5 shows B_2 , $\mathcal{J}(B_2)$, and $\mathcal{J}(B_2) / S_2$. This is the first nontrivial case, but the solution is easily obtained. B_2 is the square generated by two atoms. The ideals in $\mathcal{J}(B_2)$ are the first 6 entries in Table 9.1. In this case, one rank (rank 2) of $\mathcal{J}(B_2)$ has more than one ideal. By inspection of the shapes of the two ideals of rank 2, we see that they are isomorphic. Thus these two ideals are reduced to one class with weight 2 in $\mathcal{J}(B_2) / S_2$. The weight of each vertex in $\mathcal{J}(B_2) / S_2$ is shown outside, and adjacent to, the vertex. The number inside the vertex is the row number of the class representative from Table 9.1.

A normal flow solution is given by the flows on the edges of $\mathcal{J}(B_2) / S_2$. These flows are only valid between adjacent pairs of ranks, with an implicit start vertex s and terminal vertex t . Each pair of ranks has its own independent constraint, given by the product of the Whitney numbers of the two ranks. This product is 1 between ranks 0 and 1 and also between ranks 3 and 4, but is 2 between ranks 1 and 2 and also between ranks 2 and 3. As discussed in Chapter 5, this is a requirement on the normal flow problem for the respective adjacent ranks. Thus the individual flow solutions do not lead to a total flow on $\mathcal{J}(B_2) / S_2$ (although they would if the flows were normalized) because flow conservation would not be preserved. Since each flow problem is a chain, the flow solution for each pair of ranks is trivial.

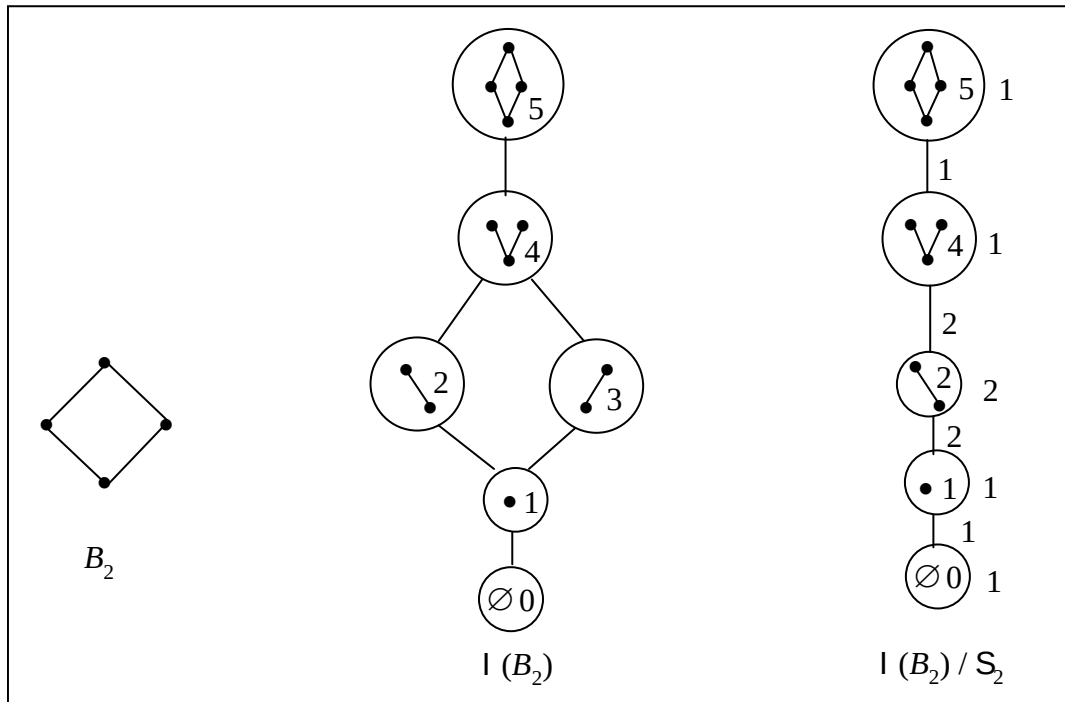


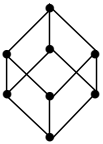
Figure 9.5. Normal Flow Solution for $\mathcal{J}(B_2)$

There are two symmetric chain decompositions of $\mathcal{J}(B_2)$, obtained by picking either ideal of rank 2 as one chain and the rest of $\mathcal{J}(B_2)$ as the other chain. A computer solution to the normal flow problem for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 6$, is given in Appendices C, E, and F.

Figure 9.6 shows B_3 , $\mathcal{J}(B_3)$, and $\mathcal{J}(B_3) / S_3$. B_3 is the familiar Boolean cube generated by three atoms. The ideals in $\mathcal{J}(B_3)$ are the first 20 entries in Table 9.1. It is straightforward to determine the equivalence classes that constitute $\mathcal{J}(B_3) / S_3$ by inspection of the shapes of the ideals in $\mathcal{J}(B_3)$. Doing so, we see that this is the first case in which $\mathcal{J}(B_k) / S_k$ is not a chain. The weight of each vertex in $\mathcal{J}(B_3) / S_3$ is shown outside, and adjacent to, the vertex. The number inside the vertex is the row number of the class representative from Table 9.1.

A normal flow solution is given by the flows on the edges of $\mathcal{J}(B_3) / S_3$. As before, these flows are only valid between adjacent pairs of ranks, with an implicit start vertex s and terminal vertex t , and must be normalized to form a total flow on $\mathcal{J}(B_3) / S_3$.

The Whitney number of the middle rank of $\mathcal{J}(B_3) / S_3$ is the sum of the weights on the two middle rank vertices. The Whitney number of each rank except for the middle rank is the weight on the only vertex of that rank. The product of the respective Whitney numbers forms the normal flow constraint for each pair of adjacent ranks. The flows are straightforward for the two cases involving the middle rank and trivial in all other cases. A computer solution to the normal flow problem for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 6$, is given in Appendices C, E, and F.



136

Table 9.2. A Symmetric Chain Decomposition of $\mathcal{J}(B_3)$

Rank				
8	19			
7	18			
6	13	16	17	
5	10	15	12	
4	5	14	11	9
3	4	8	7	
2	2	3	6	
1	1			
0	0			

From Table 9.1, $\mathcal{J}(B_4)$ has 168 ideals and so is impractical to draw. Nonetheless, it is feasible to hand-calculate $\mathcal{J}(B_4) / S_4$ and solve the normal flow problem on the lower half of the resulting Hasse diagram. Figure 9.7 shows B_4 and $\mathcal{J}(B_4) / S_4$. Although not obvious from Figure 9.7, B_4 is the 4-dimensional hypercube.

The weight of each vertex in the lower half of $\mathcal{J}(B_4) / S_4$ is shown outside, and adjacent to, the vertex. The number inside the vertex is the row number of the class representative from Table 9.1. A normal flow solution is given by the flows on the edges of $\mathcal{J}(B_4) / S_4$. As before, these flows are only valid between adjacent pairs of ranks, with an implicit start vertex s and terminal vertex t , and must be normalized to form a total flow on $\mathcal{J}(B_4) / S_4$. The Whitney numbers are the sums of the weights on the vertices of the corresponding rank. The product of the respective Whitney numbers forms the normal flow constraint for each pair of adjacent ranks. A computer solution to the normal flow problem for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 6$, is given in Appendices C, E, and F. The hand calculation of the vertices, edges and flows on $\mathcal{J}(B_4) / S_4$ will be discussed next.

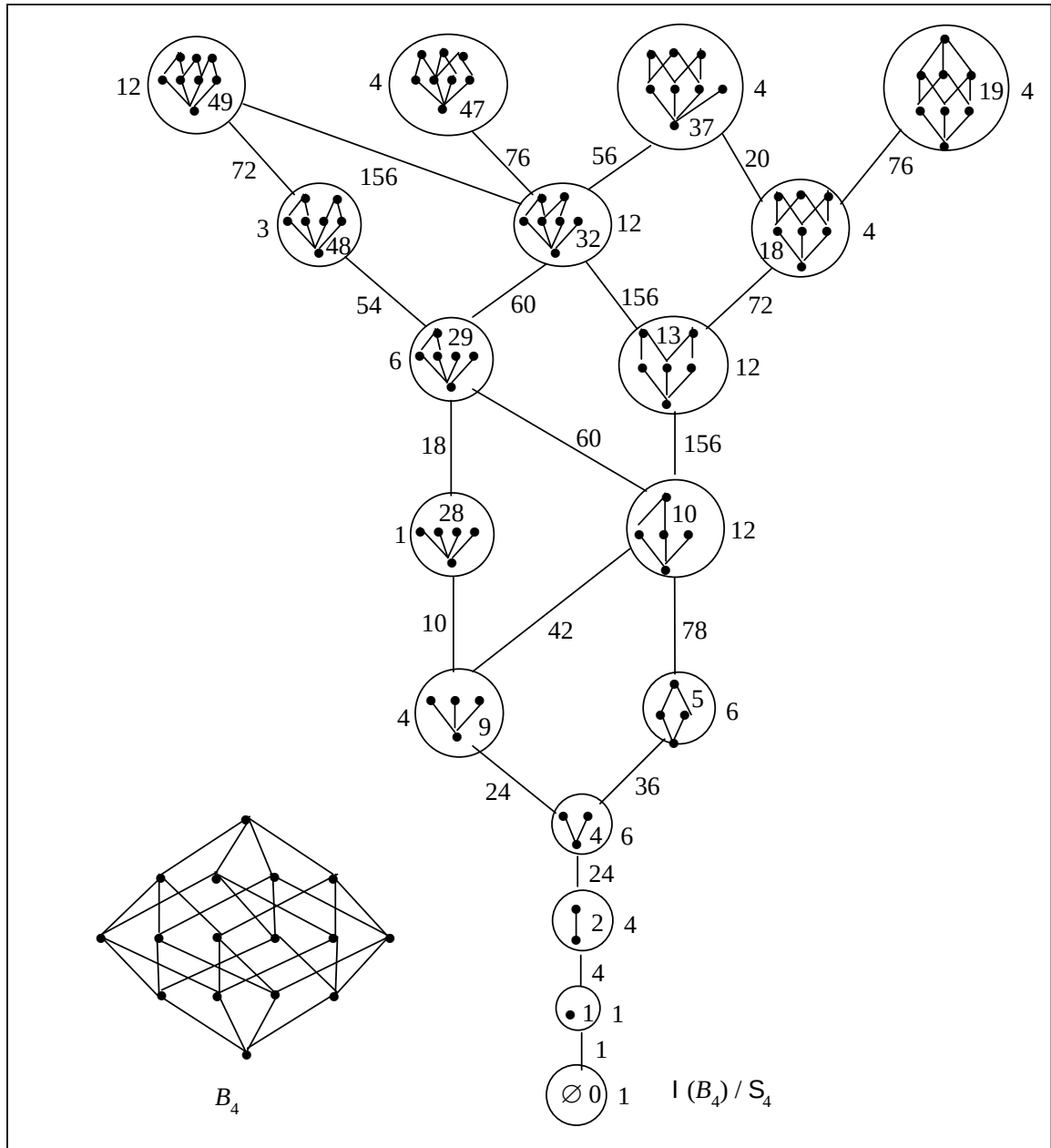


Figure 9.7. Normal Flow Solution for $\mathcal{J}(B_4) / S_4$

The only ideals of rank 0 and one in $\mathcal{J}(B_4)$ are, respectively, the null ideal and the singleton set consisting of the bottom element of B_4 , and so the respective Whitney numbers are each 1. There is an edge with flow 1 between these equivalence classes.

The only way to get from an ideal in rank 1 of $\mathcal{J}(B_4)$ to any ideal in rank 2 is to add an atom, which can be done in four different ways. Thus there is one vertex in rank 2 of $\mathcal{J}(B_4) / S_4$. It has Whitney number 4 and an edge coming in from vertex (equivalence class) 1 in rank 1. The flow on this edge is trivially 4.

Since each vertex of rank 2 in B_4 has two predecessor atoms, the only way to get from an ideal in rank 2 of $\mathcal{J}(B_4)$ to an ideal in rank 3 of $\mathcal{J}(B_4)$ is to add in another atom. The resulting ideal can be chosen in $\binom{4}{2} = 6$ ways. Thus there is one vertex (equivalence class 4) in rank 3 of $\mathcal{J}(B_4) / S_4$. It has Whitney number 6 and an edge coming in from vertex (equivalence class) 2 in rank 2. The flow on this edge is 24, the product of the weights of the vertices in ranks 2 and 3.

Since vertex (equivalence class) 4 consists of the bottom element and two atoms of B_4 , there are two ways to build an ideal of rank 4; join the two atoms in the equivalence class or add a third atom. There is one way to join the existing atoms for a total of 6 ways to have a resulting principal ideal of height 2. There are $\binom{4}{3} = 4$ ways to have three of the four atoms and no vertices of height greater than 1 in B_4 . There is thus an edge from equivalence class 4 to each of the equivalence classes 5 and 9. The Whitney numbers are 6 for rank 3 and $6 + 4 = 10$ for rank 4, so the normal flow must equal $6 \times 10 = 60$. Equivalence class 5 requires a flow of $6 \times 6 = 36$ from equivalence class 4, and equivalence class 9 requires a flow of $4 \times 6 = 24$ from equivalence class 4.

Since equivalence class 5 is a principal ideal, the only possible way to get an ideal of rank 5 is to add another atom. This can be done in two ways, so the resulting

equivalence class 10 has size $6 \times 2 = 12$. Equivalence class 9 can be extended in two ways to an ideal of rank 5. One way is to join two of the existing atoms. This gives the same equivalence class, 10, that can be built from equivalence class 5. The other way is to add in the remaining atom, which can only be done one way, resulting in equivalence class 28. The Whitney numbers are 10 and 13, requiring a flow of 130. All $6 \times (12 + 1) = 78$ of the flow out of equivalence class 5 must go to equivalence class 12. Similarly, all $1 \times (4 + 6) = 10$ of the flow into equivalence class 28 must come from equivalence class 9. The remaining flow out of equivalence class 9 and into equivalence class 10 must equal $4 \times (1 + 12) - 10 = 42 = 12 \times (4 + 6) - 78$.

Equivalence class 28 can only be extended by joining two atoms, forming equivalence class 29. This can be done in $\binom{4}{2} = 6$ ways. Equivalence class 10 can also be extended to 29 by adding in an atom. Alternatively, 10 can be extended to equivalence class 13 by joining two atoms, one of which is necessarily already joined to the third atom in the class. There are $\binom{4}{3} = 4$ ways to choose the three atoms, and 3 ways to pick the atom that is joined with each of the other two atoms, so equivalence class 13 has weight 12. The Whitney numbers are 13 and 18, requiring a flow of $13 \times 18 = 234$. A flow of $1 \times 18 = 18$ must go from 28 to 29, and a flow of $13 \times 12 = 156$ must come from 10 to 13, and so the remaining flow of 60 must go from 10 to 29.

Equivalence class 29 has all four atoms with one joined pair of atoms, and thus can only be extended by joining another pair of atoms. There are two ways to do this. One way is to join the other two atoms (equivalence class 48). Class 48 has weight 3;

pick one atom and join it with any of the other 3, forcing the remaining two atoms to be joined with each other. The other way is to join one previously joined atom and one atom that has not been previously joined (equivalence class 32). Class 32 has size $12 = 4$ ways to pick the “orphan” atom times 3 ways to pick the atom joined twice. Class 32 can also be built from equivalence class 13 by adding in the remaining atom. Alternatively, equivalence class 13 can get to equivalence class 18 by joining the remaining two atoms in the class that have not already been joined. This class has weight 4; once the “orphan” atom has been excluded from the ideal, the remaining three atoms must be joined in all possible ways. The Whitney numbers are 18 and 19, requiring a flow of 342. Class 48 must get all 54 of its flow from class 29, and class 18 must get all 72 of its flow from class 13. The remaining 60 units of flow from class 29, and the remaining 156 units of flow from class 12 then must go into class 32.

The only way to have an ideal of rank 8 in $\mathcal{J}(B_4)$ using less than four atoms is to have a principal ideal. This is equivalence class 19, which has weight $\binom{4}{3} = 4$ and must have equivalence class 18 as its only predecessor. All other ideals of rank 8 in $\mathcal{J}(B_4)$ must contain all four atoms and three vertices of rank 2 in B_4 . The only other way to extend class 18 is to add in the other atom. This is equivalence class 37, which has weight 4 because there are four ways to choose the unpaired atom. Equivalence class 48 can only be extended by joining two atoms that are not already joined with each other. Since each atom of class 48 is joined with one other atom, the result must be equivalence class 49. The weight of this class is 12; the two atoms that are joined with each other can be chosen $\binom{4}{2} = 6$ ways, and there are 2 ways to join each of these atoms with one of

the other two atoms. This leaves equivalence class 32. This class can be extended to class 49 by joining the unpaired atom with an atom that has been joined once, or to class 37 by joining the two atoms that have each been joined with a common atom, or to equivalence class 47 by joining the unpaired atom to the atom that is already joined to the remaining two atoms. Equivalence class 47 has weight 4 because there are four ways to pick the atom that is joined with each of the other three atoms. The Whitney numbers are 19 and 24, requiring a flow of 456. Class 48 must give all 72 of its flow to class 49, and class 19 must get all 76 of its flow from class 18. Then the remaining flow of 20 out from class 18 must go to class 37. The remaining flow of 156 into class 49, the flow of 76 into class 47, and the remaining flow of 56 into class 37 must all come from class 32, accounting for its required flow of 288.

A symmetric chain decomposition of $\mathcal{J}(B_4)$ is shown in Table 9.3. Table 9.1 can be used to check the predecessor and successor relations in the chains of Table 9.3. This symmetric chain decomposition was obtained in an ad-hoc fashion, through trial and error. Nonetheless, every chain has a dual chain in the chain decomposition. There are 24 chains in the chain decomposition (as there must be, because there are 24 ideals in the middle rank). The 12 that contain a self-dual ideal of rank 8 are self-dual chains; the other 12 are not. Table 9.3 shows the dual for each chain in the chain decomposition.

This style of analysis is no longer practical for $k = 5$, which has 113 vertices in each half of $\mathcal{J}(B_5) / S_5$. The normal flow solutions for $\mathcal{J}(B_5) / S_5$ and $\mathcal{J}(B_6) / S_6$ are shown in Appendices C, E and F. The solution for $\mathcal{J}(B_6) / S_6$ was generated by running ranks 0 through 20 in one run, then adjacent pairs of ranks up to the middle rank (ranks 20 and 21

in one run, ranks 21 and 22 in one run, ..., ranks 31 and 32 in one run). Ranks 33 through 64 were not necessary to run because of the symmetry of $\mathcal{J}(B_k) / S_k$ about the middle rank (Corollary 4.16).

Note that, for each pair of ranks of $\mathcal{J}(B_k) / S_k$ for $k = 0, 1, \dots, 4$, there is exactly one solution to the normalized flow problem. One might expect that increasing values of k would increase the constraints on the flow problem, rendering a solution impossible. In fact, for $k = 5$ and $k = 6$, the opposite is true; the solutions have “slack”, as will be shown later in this chapter.

Table 9.3. A Symmetric Chain Decomposition of $\mathcal{J}(B_4)$ (1 of 2)

	1	2	3	4	5	6	7	8	9	10	11	12	Chain
	1	2	3	4	6	5	7	9	8	10	11	12	Dual
Rank													
16	167												
15	166												
14	147	161	164	165									
13	133	160	163	146	158	144							
12	83	159	162	132	153	128	139	157	143	145			
11	77	155	156	82	119	108	138	152	142	141	127	130	
10	52	154	151	81	66	107	136	118	137	140	76	80	
9	38	149	117	79	65	51	103	113	106	135	74	72	
8	19	148	95	70	60	49	102	91	50	134	63	47	
7	18	114	93	68	32	35	46	90	48	101	58	44	
6	13	109	92	55	31	16	45	89	34	98	29	41	
5	10	56	86	24	12	15	30	85	33	42	28	40	
4	5	53	84	23	9	14	11	26	27	39			
3	4	22	25	21	7	8							
2	2	3	6	20									
1	1												
0	0												

Table 9.3. A Symmetric Chain Decomposition of $\mathcal{J}(B_4)$ (2 of 2)

	13	14	15	16	17	18	19	20	21	22	23	24	Chain
	13	14	16	15	18	17	19	23	21	24	20	22	Dual
Rank													
16													
15													
14													
13													
12													
11	131												
10	129	97	125	123	126	150							
9	121	96	105	122	75	116	124						
8	120	37	100	71	64	94	115	73	78	104	111	112	
7	69	36	99	59	62	88	110						
6	67	17	43	57	61	87							
5	54												
4													
3													
2													
1													
0													

Table 9.4 shows the Whitney numbers for $\mathcal{J}(B_0)$ through $\mathcal{J}(B_6)$. Only the lower half of $\mathcal{J}(B_6)$ is shown. As can be seen from Table 9.4, each case is unimodal. By Corollary 4.16, each case is also rank symmetric, and this is reflected in Table 9.4 for $\mathcal{J}(B_0)$ through $\mathcal{J}(B_5)$. Since we have shown each case to be strongly Sperner, each case is thus Peck. The Dedekind numbers are also shown for comparison to their known values given in Appendix A. The Dedekind numbers are the sum of the Whitney numbers; for $\mathcal{J}(B_6)$, this may be calculated as the Whitney number of the middle rank plus twice the sum of the Whitney numbers of the other ranks in the lower half of $\mathcal{J}(B_6)$.

Table 9.4. Whitney Numbers for $\mathcal{J}(B_0)$ through $\mathcal{J}(B_6)$

Rank	$\mathcal{J}(B_0)$	$\mathcal{J}(B_1)$	$\mathcal{J}(B_2)$	$\mathcal{J}(B_3)$	$\mathcal{J}(B_4)$	$\mathcal{J}(B_5)$	$\mathcal{J}(B_6)$ lower half
32						1	492,288
31						1	484,376
30						5	463,065
29						10	430,280
28						20	390,050
27						35	345,690
26						61	300,351
25						95	256,200
24						155	215,035
23						215	177,530
22						310	144,505
21						387	115,782
20						470	91,430
19						530	71,000
18						580	54,195
17						605	40,488
16					1	621	29,566
15					1	605	20,950
14					4	580	14,345
13					6	530	9,415
12					10	470	5,913
11					13	387	3,512
10					18	310	2,045
9					19	215	1,110
8				1	24	155	605
7				1	19	95	306
6				3	18	61	156
5				3	13	35	75
4			1	4	10	20	35
3			1	3	6	10	15
2		1	2	3	4	5	6
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1
Dedekind number	2	3	6	20	168	7,581	7,828,354

We conclude this chapter with analyses of the slack in flow calculations, execution times, and retries required by the ideal exchange algorithm under the conditions studied in this research.

As observed earlier in this chapter, there is no slack in the assignment of flows to show the normality of $\mathcal{J}(B_k) / S_k$ for $k = 1, 2, 3$, or 4 ; that is, there is exactly one assignment of normal flows between each pair of adjacent ranks for any of these cases. This is no longer the case for $k = 5$ or $k = 6$. To see this, let us adopt the following informal definition of “slack”. The motivation for the definition will follow.

Definition 9.3: A normal flow between two adjacent ranks of a poset has *slack* if there is at least one unused edge; i.e., an edge whose flow is 0. The *slack between adjacent ranks* in a flow between the ranks is the number of 0 edges in the flow.

To motivate this definition, it is useful to examine the occurrence of a 0 flow that occurs between the lowest pairs of adjacent ranks (ranks 7 and 8) of $\mathcal{J}(B_5) / S_5$. Figure 9.8 shows the flow calculated by the MPM algorithm between ranks 7 and 8 of $\mathcal{J}(B_5) / S_5$.

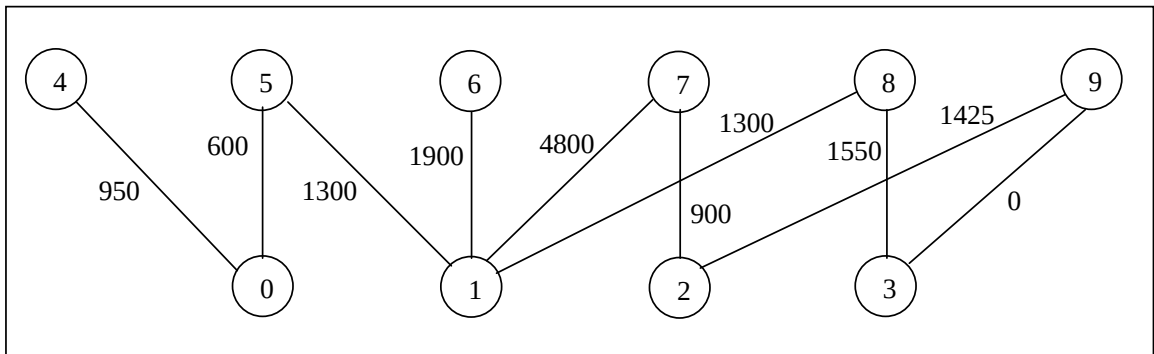


Figure 9.8. Slack in the Flow Between Ranks 7 and 8 in $\mathcal{J}(B_5) / S_5$

There is one edge with 0 flow. It is on the cycle $1 - 7 - 2 - 9 - 3 - 8 - 1$. In this cycle (necessarily of even length), we say that edges $1 - 7$, $2 - 9$, and $3 - 8$ are in the *same direction*. Similarly, edges $7 - 2$, $9 - 3$, and $8 - 1$ are in the same direction as each other but in the *opposite direction* of the other edges. By adding 1 unit of flow to each edge in the same direction of the cycle as the 0 edge, and subtracting 1 unit of flow from each edge in the opposite direction of the cycle, a new normal flow solution is obtained with the same total flow and no 0 edges.

Proposition 9.4: Suppose there is a cycle in a bipartite network. If there is a network flow such that each edge in one direction of the cycle has positive flow, there is an equivalent network flow (with the same total flow and the same flow into and out of each vertex) but different flows on some edges.

Proof: Subtract 1 unit of flow from each edge in a direction of the cycle that has all positive flows. Add 1 unit of flow to each edge in the opposite direction of the cycle. Each vertex is on exactly one edge of the cycle in each direction, so flow conservation is preserved. Since the cycle has even length, the number of flow units added in one direction is equal to the number of flow units subtracted in the other direction.

Corollary 9.5: Suppose there is a cycle in a bipartite network. Further suppose there is a network flow with one or more 0 edges in the same direction of the cycle but each edge in the opposite direction of the cycle has a flow of at least 2. Then there is an equivalent flow with no 0 edges on the cycle.

The point of Proposition 9.4 is that the existence of cycles in the bipartite flow problem on adjacent ranks of $\mathcal{J}(B_k) / S_k$ for $k \geq 5$ provides a “degree of freedom” that is not present for $k \leq 4$, where the flow on each edge is constrained to one specific value.

Corollary 9.5 focuses on cycles with 0 edges, which are easier to find. Clearly, each vertex on any 0 edge must each be on at least one other edge with a positive flow. Inspection of the output file for $\mathcal{J}(B_5) / S_5$ shows that every 0 edge is on a cycle that satisfies the conditions of Corollary 9.5. This suggests that the 0 edges could have been used in the normal flow solutions, and so they are indicators of an additional degree of freedom in the normal flow solutions.

With this background, we now look at the slack by rank and Whitney number in the flow solutions for $\mathcal{J}(B_5) / S_5$ and $\mathcal{J}(B_6) / S_6$. Figure 9.9 shows the slack by rank for $k = 5$. Figure 9.10 shows the slack by Whitney number for $k = 5$. Figure 9.11 shows the slack by rank for $k = 6$. Figure 9.12 shows the slack by Whitney number for $k = 6$. In each figure, the rank is the higher of the two ranks in the corresponding bipartite flow problem. The results were symmetric, as expected, for $k = 5$, so only the flow problems for the lower half of $\mathcal{J}(B_5) / S_5$ and $\mathcal{J}(B_6) / S_6$ are shown in each figure.

Inspection of the figures shows that the slack values are monotone increasing by both k and rank (up to the middle rank). Qualitatively, this suggests that the degree of freedom increase with the size of the bipartite normal flow problems, and so it is a plausible conjecture that normality of $\mathcal{J}(B_k) / S_k$ holds in general.

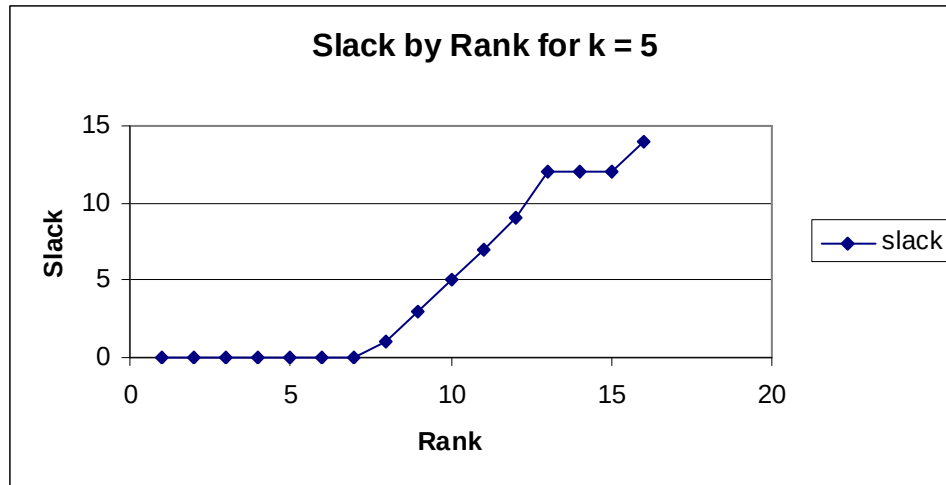


Figure 9.9. Slack by Rank for $k = 5$

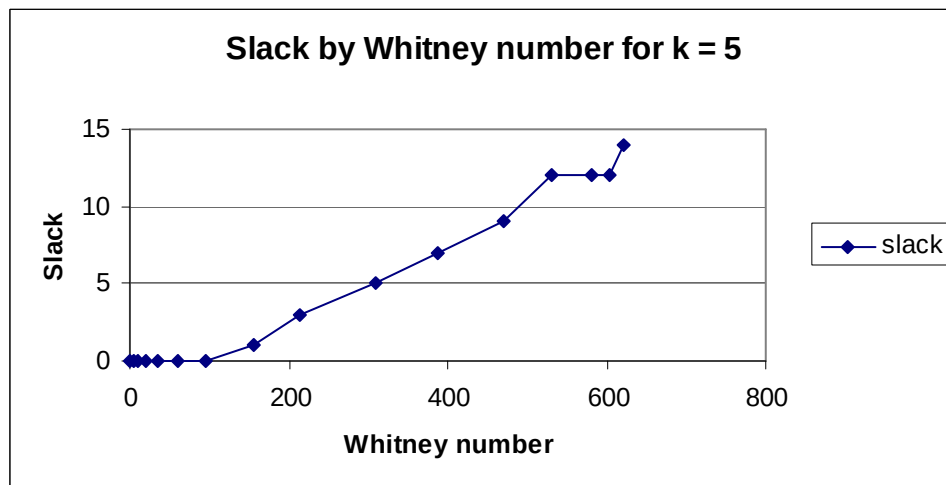


Figure 9.10. Slack by Whitney number for $k = 5$

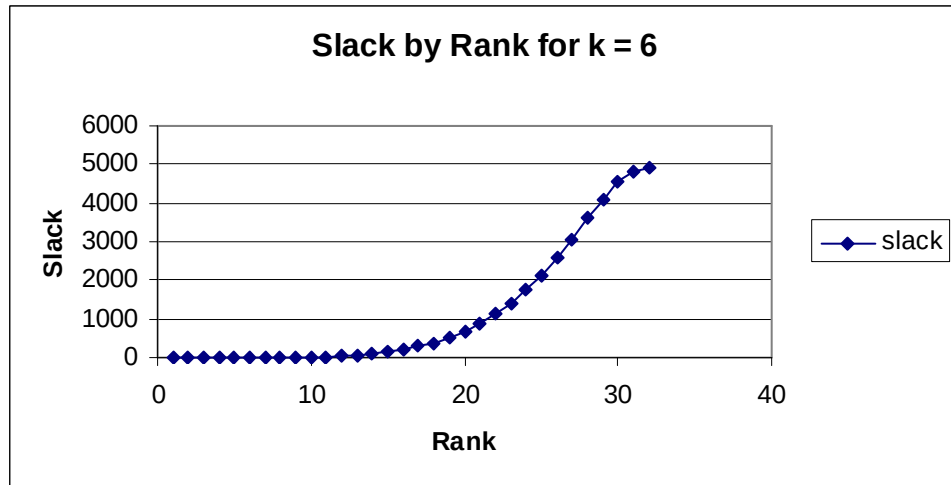


Figure 9.11. Slack by Rank for $k = 6$

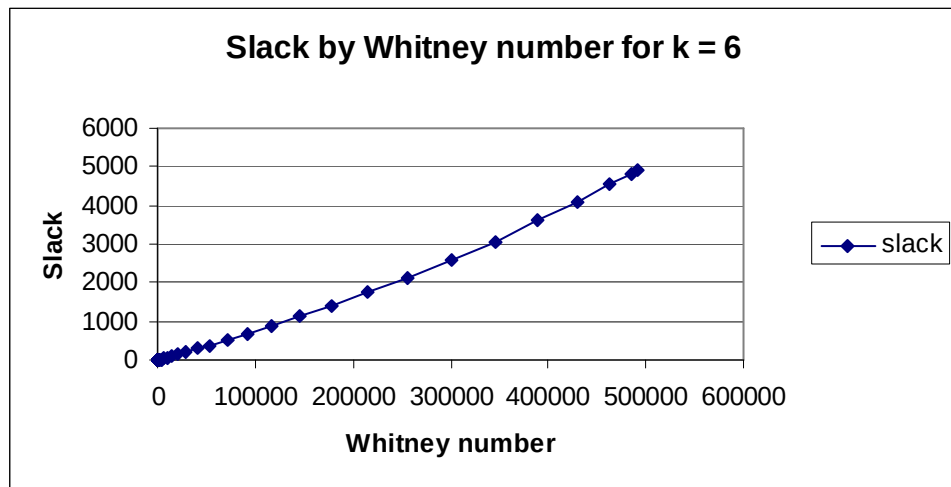


Figure 9.12. Slack by Whitney number for $k = 6$

The execution times under various conditions will be analyzed next. The quotient poset generation, flow calculation, and total execution times for $\mathcal{J}(B_k) / S_k$, $k = 1, \dots, 6$, using the ideal exchange algorithm are shown to the nearest second in Table 9.5.

Table 9.5. Execution Time for $\mathcal{J}(B_k) / S_k$ Using Ideal Exchange Algorithm

k	Quotient Poset Generation (Seconds)	Flow Calculation (Seconds)	Total
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	6	0	6
6 ¹	1,682,388	65	1,682,453

¹Lower half of $\mathcal{J}(B_6) / S_6$

Quotient poset generation time predominates. The near double exponential growth becomes apparent for $k = 6$. The flow calculation times are less than 0.5 seconds for $k \leq 5$, and the quotient poset generation times are less than 0.5 seconds for $k \leq 4$. The flow calculation time for the lower half of $\mathcal{J}(B_6) / S_6$ is just over a minute, whereas the quotient poset generation time for the lower half of $\mathcal{J}(B_6) / S_6$ is almost 19 ½ days.

The breakdown by rank of the execution times for the lower half of $\mathcal{J}(B_6) / S_6$ is shown in Figure 9.13.

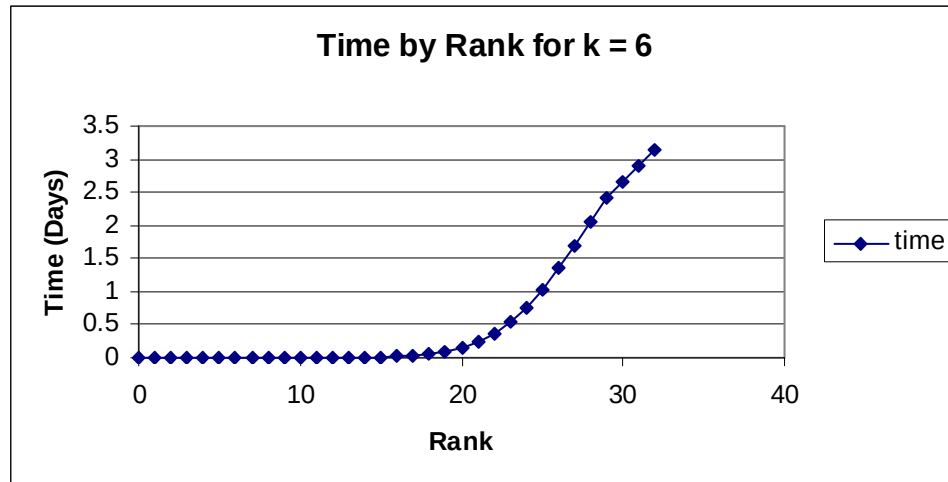


Figure 9.13. Total execution time by rank for $k = 6$

Figure 9.14 shows the same data converted to a logarithmic scale for those ranks with a total execution time of at least one second. The resulting graph is not quite linear, suggesting that the timing behavior of Figure 9.13 is not quite exponential.

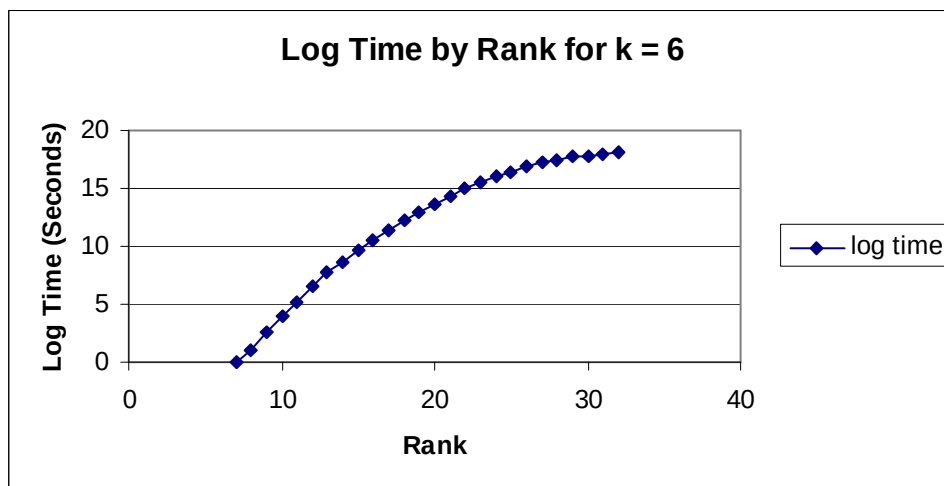


Figure 9.14. Log execution time by rank for $k = 6$

It is interesting to analyze the effect of using the quotient poset $\mathcal{I}(B_k) / S_k$ instead of the poset $\mathcal{I}(B_k)$ for the normality test.

The use of $\mathcal{I}(B_k) / S_k$ instead of $\mathcal{I}(B_k)$ has three effects on time and storage. The size of $\mathcal{I}(B_k) / S_k$ is smaller than the size of $\mathcal{I}(B_k)$ by $O(k!)$, since each equivalence class under the action of S_k can be as large as $k!$. As a result, the flow calculation is faster on $\mathcal{I}(B_k) / S_k$. The poset generation time, however, is slower by $O(k!)$ using $\mathcal{I}(B_k) / S_k$ instead of $\mathcal{I}(B_k)$. Each ideal must be generated in either case. Using $\mathcal{I}(B_k) / S_k$, each ideal must be put into the appropriate equivalence class under the action of S_k , requiring $O(k!)$ isomorphism tests per ideal.

Thus, for the poset generation portion of the normality test, there is a classic trade-off of $O(k!)$ more storage required without equivalence classes vs. $O(k!)$ more computation time using equivalence classes. However, as shown in Table 9.6, the flow calculation time using the smaller quotient poset network is faster by a sufficiently large amount that using the quotient poset is better in terms of both time and storage than using the ideal poset. It should be noted that, for $k = 6$, the memory requirements using the ideal poset became prohibitive, so the data are only shown for ranks 1 through 13. (As before, the rank entry is the higher of the two ranks in the bipartite network.

Table 9.6. Execution Time by Rank for $\mathcal{I}(B_6) / S_6$ and $\mathcal{I}(B_6)$

Upper Rank	$\mathcal{I}(B_6) / S_6$ Total Time (Seconds)	$\mathcal{I}(B_6)$ Total Time (Seconds)
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	1	1
8	2	4
9	6	10
10	16	35
11	35	106
12	96	317
13	223	885

As seen from Table 9.6, the computation time using the quotient poset is always faster than the computation time using the ideal poset over the ranks considered. In fact, the difference grows at a rate faster than linear, as shown in Figure 9.15.

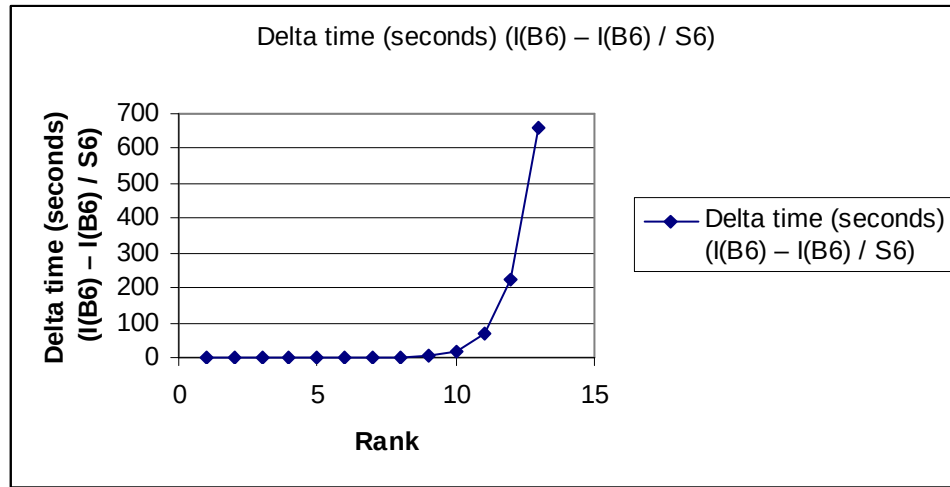


Figure 9.15. Delta Time (Seconds) ($\mathcal{I}(B_6) - \mathcal{I}(B_6) / S_6$)

As observed earlier, quotient poset generation dominates the computation time for $\mathcal{I}(B_6) / S_6$. For $\mathcal{I}(B_6)$, flow calculation dominates the computation time. Figure 9.16 compares the ideal generation time using $\mathcal{I}(B_6) / S_6$ and $\mathcal{I}(B_6)$. Figure 9.17 compares the flow calculation time using $\mathcal{I}(B_6) / S_6$ and $\mathcal{I}(B_6)$.

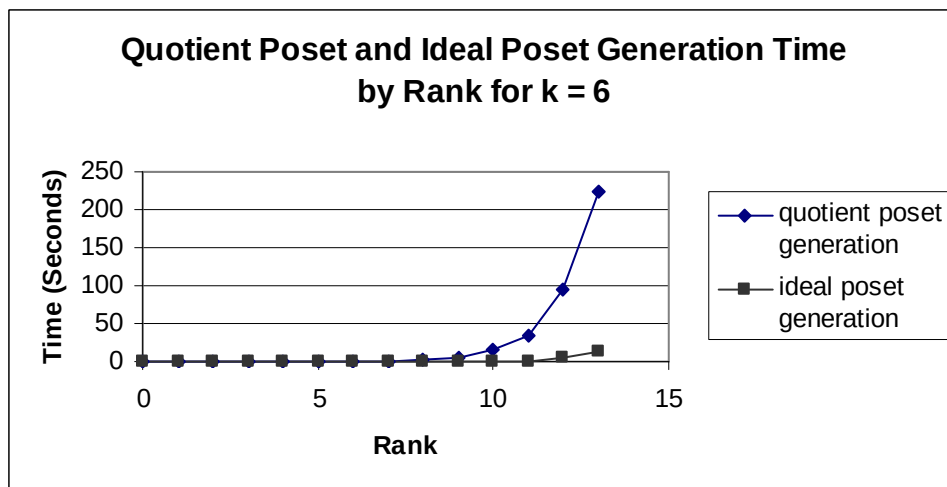


Figure 9.16. Quotient Poset and Ideal Poset Generation Time by Rank

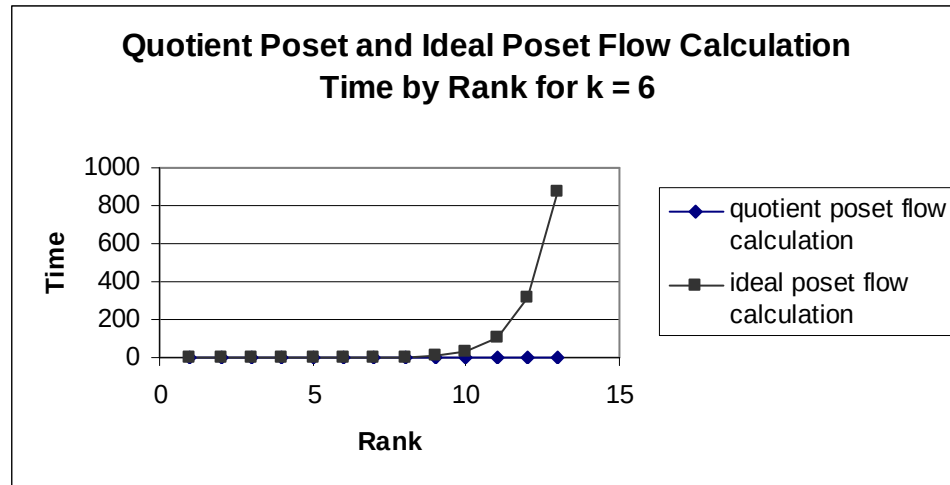


Figure 9.17. Quotient Poset and Ideal Poset Flow Calculation Time by Rank

The remaining execution time comparison involves the choice of the ideal exchange algorithm or the selective ideal generation algorithm. The ideal exchange algorithm bypasses ideals that are not of interest, thereby saving time when not every ideal is of interest. However, the ideal exchange algorithm sometimes requires retries as discussed in Chapter 7. This adds to the execution time.

Figure 9.18 shows retries by rank for $k = 5$ using the ideal exchange algorithm.

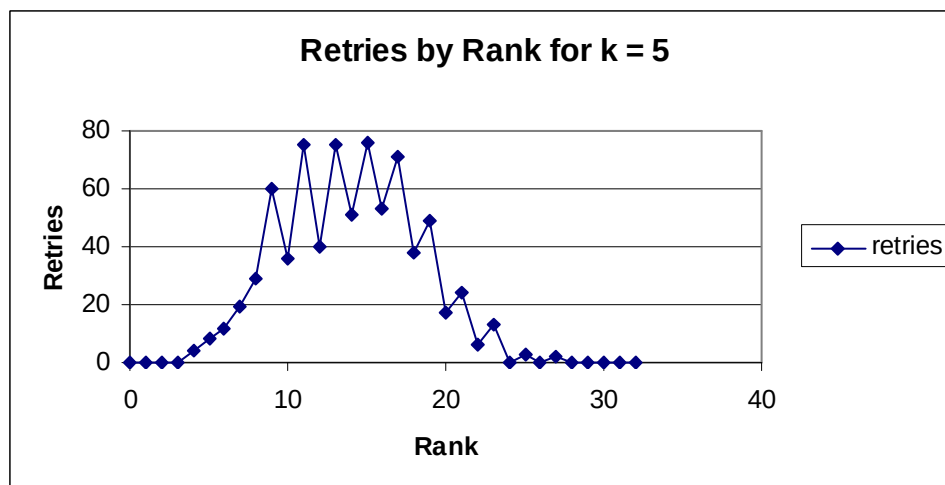


Figure 9.18. Retries by Rank for $k = 5$ Using Ideal Exchange Algorithm

Interestingly, the retries are not monotone on either the upper half or lower half of the ideal poset and are not symmetric between the upper half and lower half of the ideal poset. This behavior is especially clear in Figure 9.19, which shows the retries by Whitney number for $k = 5$ using the ideal exchange algorithm. The Whitney number is the number of ideals in the rank of interest. The Whitney numbers repeat in the upper half of the ideal poset. In each case in which the retry count is nonzero in the lower half, the retry count in the upper half is smaller (and sometimes 0). This is perhaps because the ideal exchange algorithm first finds a vertex to add from the complement. The available vertices increase by rank in the lower half and decrease by rank in the upper half. The reason for the non-monotonicity is unclear and may have structural significance pertinent to the general solution of the problem.

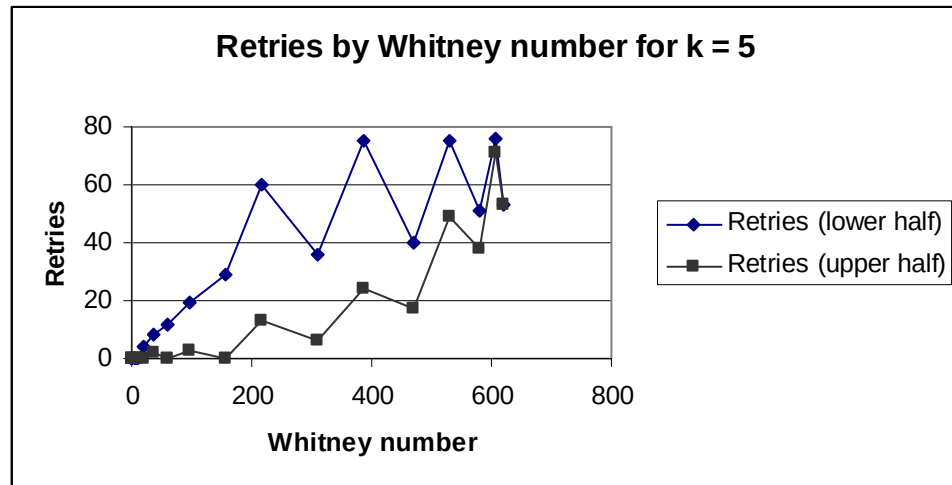


Figure 9.19. Retries by Whitney Number for $k = 5$ Using Ideal Exchange Algorithm

Figure 9.20 shows the retries by rank for $k = 6$ using the ideal exchange algorithm. Figure 9.21 shows the retries by Whitney number for $k = 6$. Only the lower

half of the quotient poset was generated. In hindsight, a slight improvement in running time could be expected if the upper half were used instead.

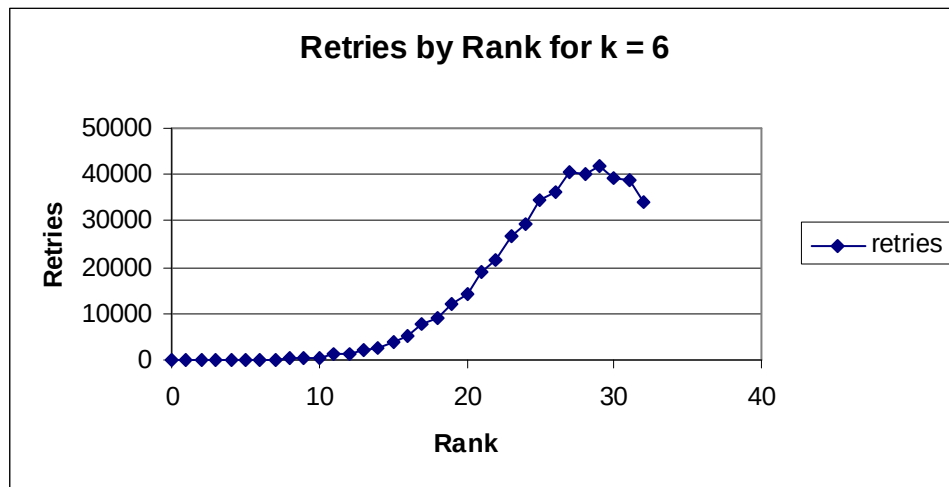


Figure 9.20. Retries by Rank for $k = 6$ Using Ideal Exchange Algorithm

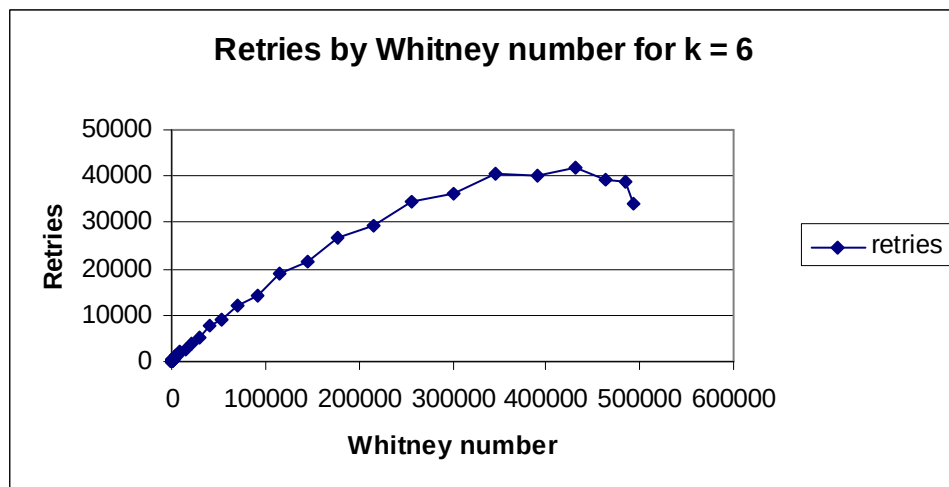


Figure 9.21. Retries by Whitney Number for $k = 6$ Using Ideal Exchange Algorithm

We conclude this chapter with Table 9.7, which compares the worst-case execution times using the ideal exchange algorithm and the selective ideal generation algorithm. Ranks 32 and 33 were chosen for the selective ideal generation algorithm because the behavior of this algorithm is better on the upper half of the ideal poset.

Table 9.7. Execution Time by Algorithm for $\mathcal{I}(B_6) / S_6$ (Middle Rank)

Algorithm	Ranks	Quotient Poset Generation Time (Seconds)
exchange	31-32	520178 (6 days 0 hours 29 minutes 38 seconds)
selective	32-33	529646 (6 days 3 hours 7 minutes 26 seconds)

The execution time improvement resulting from using the ideal exchange algorithm instead of the selective ideal generation algorithm was 2 hours 37 minutes 48 seconds over a 6-day period, or about 1.8%.

CHAPTER 10: FUTURE DIRECTIONS

Several questions regarding Sperner properties of the ideals of a Boolean lattice and related posets remain open. Due to the symmetry of B_k and the normality test results, with increasing slack for increasing k as discussed in Chapter 9, this author makes the following conjectures that have yet to be proved for all k :

$\mathcal{J}(B_k)$ is unimodal.

$\mathcal{J}(B_k)$ is RUSS. If true, then $\mathcal{J}(B_k)$ is Peck (Corollary 4.14) and strongly Sperner.

$\mathcal{J}(B_k)$ has a symmetric chain decomposition. If true, $\mathcal{J}(B_k)$ is Peck [Engel 1997] and therefore strongly Sperner by definition.

$\mathcal{J}(B_k)$ is normal. If true, $\mathcal{J}(B_k)$ is strongly Sperner [Harper 1974].

On the other hand, while the evidence that $\mathcal{J}(B_k)$ is normal for all k is persuasive based on the results presented in Chapter 9, Π_n , the lattice of partitions of an n -set, is normal for $n < 20$ but not for $n = 20$ [Harper 1974]. This serves as a reminder that evidence such as that in Chapter 9 does not constitute a proof that $\mathcal{J}(B_k)$ is normal, or even Sperner, for all k .

The analysis of this paper can, in principle, be extended beyond $k = 6$, although in practice, due to the near double exponential growth of the size of $\mathcal{J}(B_k)$, such analysis quickly becomes impractical. Recall from Chapter 5 that the MPM algorithm is $O(|V|^3)$. For $k = 6$, the Dedekind number $D_6 \approx 7 \times 10^6$ (see Appendix A), so $|\mathcal{J}(B_6) / S_6|$ is about $(7 \times 10^6) / 6! \approx 10^4$, and thus the MPM algorithm is $O(10^{12})$. For $k = 7$, $|\mathcal{J}(B_7) / S_7|$ is approximately $(2 \times 10^{12}) / 7! \approx 4 \times 10^8$, so the MPM algorithm is $O(6.4 \times 10^{25})$.

In practice, however, the MPM algorithm runs much faster on this problem than its theoretical bound would indicate. As shown in Chapter 9, the calculation times are dominated by the generation of $\mathcal{J}(B_k) / S_k$. If this holds true for higher values of k , the execution time for $k = 7$ is mainly the time to generate $\mathcal{J}(B_7) / S_7$, which is approximately $O((2 \times 10^{12}) \times 7!) \approx 10^{16}$. By comparison, $\mathcal{J}(B_6) / S_6$ is $O((7 \times 10^6) \times 6!) \approx 5 \times 10^9$. Thus $\mathcal{J}(B_7) / S_7$ could take 2×10^6 times as long to solve as $\mathcal{J}(B_6) / S_6$. Since $\mathcal{J}(B_6) / S_6$ takes about a week running in parallel, $\mathcal{J}(B_7) / S_7$ could take on the order of 40,000 years!

The categorical imperative [Mac Lane 1998], [Goguen 1991] and the categorical approach of L. H. Harper to combinatorics provide a potentially fruitful alternative approach to proving Sperner properties of posets. L. H. Harper has used category theory to solve several combinatorial problems.

[Birkhoff 1967] studied the free distributive lattice on n generators. [Stanley 1991] showed that $\mathcal{J}(B_k)$ is Sperner for $k \leq 4$. Sperner's theorem and many extensions are based on representable functors that are naturally promotable from **Set** to **Poset**. For example, the Boolean lattice is the range of the functor that takes each finite set to its power set. This is representable by $\mathcal{P}(A) \cong \text{Hom}_{\text{Set}}(A, \mathcal{N}_2)$; $B \subseteq A$ iff $\phi_B \in \text{Hom}_{\text{Set}}(A, \mathcal{N}_2)$, where $B = \{x \in A \mid \phi_B(x) = 1\}$. This functor is promotable from **Set** to **Poset** in the standard fashion: $f \leq g$ iff $(\forall x)\{f(x) \leq g(x)\}$.

[Harper 1982] showed that intertwining operators are the morphisms of the category **RUSS**. [Harper 2004] showed that stabilization morphisms are useful in optimizing measures such as wirelength and bandwidth in circuit card layouts. A natural stabilization $\text{Stab} : \mathcal{J}(B_k) \times \mathcal{J}(B_k) \rightarrow \mathcal{J}(B_{k+1})$ was considered as a candidate **RUSS**

morphism. If $Stab$ were to be a **RUSS** morphism (an intertwining operator with additional properties to be described in the next paragraph), then $\mathcal{J}(B_k)$ would be RUSS (and therefore strongly Sperner) for all k . This is because we have shown $\mathcal{J}(B_k)$ to be RUSS for $k \leq 6$, and the **RUSS** product theorem [Proctor, Saks and Sturtevant 1980] shows that $\mathcal{J}(B_k) \times \mathcal{J}(B_k)$ is RUSS whenever $\mathcal{J}(B_k)$ is RUSS. The missing piece is to show that $Stab : \mathcal{J}(B_k) \times \mathcal{J}(B_k) \rightarrow \mathcal{J}(B_{k+1})$ is an intertwining operator.

After some preliminary discussion, we will see that $Stab$ is not a **RUSS** morphism. Following [Proctor 1982J], associate to each rank i of a graded poset P a complex vector space P_i freely generated by the elements of rank i in P , and a vector space $P = \bigoplus_{i=0}^r P_i$ for the entire poset, where r is the maximum rank. A *raising operator* ρ is a linear transformation on P such that, for each $v \in P$, $\rho(v)$ is a linear combination of the elements of P that cover v . Following [Harper 1982], given posets P and P' with respective raising operators ρ and ρ' , an *intertwining operator* $\Psi : P \rightarrow P'$ is a linear transformation such that $\Psi \bullet \rho = \rho' \bullet \Psi$; i.e., the diagram of Figure 10.1 commutes.

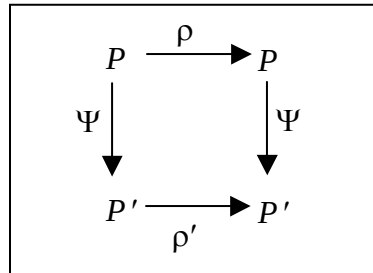


Figure 10.1. Intertwining Operator Commutative Diagram

Thus an intertwining operator Ψ takes raising operators in P to raising operators in P' . We levy some additional constraints on Ψ . We require Ψ to be a digraph morphism on the Hasse diagrams. We also require a weight condition to be satisfied. Let $\rho(u, v)$ be the coefficient of v in the expansion of $\rho(u)$ (i.e., the weight on the edge $u \rightarrow v$ in the Hasse diagram of P). Then, for any $x' \rightarrow y'$ in P' and for any x in the pre-image of x' , the sum of the weights on the edges $x \rightarrow y$ emanating from x and mapped to $x' \rightarrow y'$ under Ψ is a constant (i.e., independent of the choice of x in the pre-image of x'). This means, for example, if $x_1 \rightarrow y_1$, $x_1 \rightarrow y_2$, $x_2 \rightarrow y_2$, and $x_2 \rightarrow y_3$ are the pre-images of $x \rightarrow y$ under Ψ , then $\rho(x_1, y_1) + \rho(x_1, y_2) = \rho(x_2, y_2) + \rho(x_2, y_3)$. (There can also be weights imposed by Ψ on the vertices, but this is not essential to the following.) Any map Ψ that satisfies all of the foregoing conditions is a **RUSS morphism**; [Harper 1982] shows that such a map Ψ is indeed a morphism in the category **RUSS**.

We are now ready to define our candidate morphism *Stab* and show that it does not have the properties required of a **RUSS morphism**. Consider $\mathcal{J}(B_1) \times \mathcal{J}(B_1) \rightarrow \mathcal{J}(B_2)$ as shown in Figure 10.2. Each ideal is principal. To simplify notation in $\mathcal{J}(B_1) \times \mathcal{J}(B_1)$, each ideal is represented in $\mathcal{J}(B_1)$ by its maximum vertex. At each level, isomorphic vertices of $\mathcal{J}(B_1) \times \mathcal{J}(B_1)$ are “compressed” into a class representative in $\mathcal{J}(B_2)$. To clarify the correspondence between edges in $\mathcal{J}(B_1) \times \mathcal{J}(B_1)$ and edges in $\mathcal{J}(B_2)$, each member of an equivalence class of edges (with isomorphic tails and isomorphic heads) is given the same weight, unique to the class. The image of each edge in $\mathcal{J}(B_2)$ has this

common weight multiplied by the number of edges in the pre-image emanating from a common vertex (2 in two cases, 1 in all other cases).

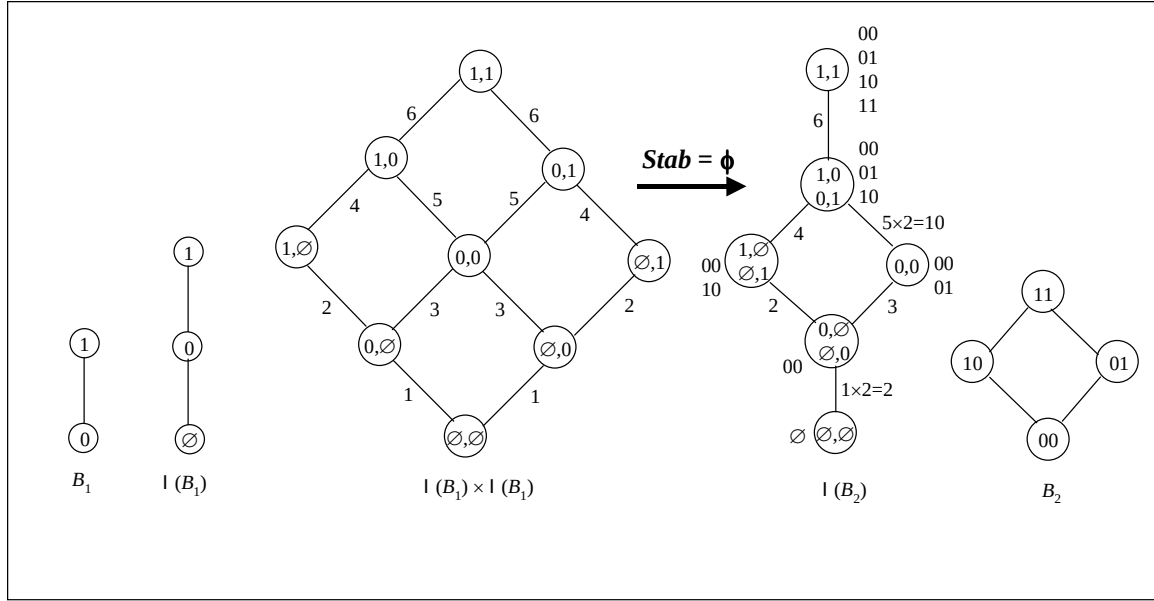


Figure 10.2. Intertwining Operator $Stab : \mathcal{I}(B_1) \times \mathcal{I}(B_1) \rightarrow \mathcal{I}(B_2)$

By inspection of Figure 10.2, this “works”; i.e., $Stab : \mathcal{I}(B_1) \times \mathcal{I}(B_1) \rightarrow \mathcal{I}(B_2)$ is an intertwining operator. A fundamental reason why is that the pre-images of any edge in $\mathcal{I}(B_2)$ are all isomorphic. Unfortunately, this key property does not generalize. To gain greater insight into this, it is enlightening to examine $Stab$ more closely. The element (class representative) (i, j) shown in each vertex of $\mathcal{I}(B_2)$ is the argument of $Stab$, and the ideal shown outside each vertex of $\mathcal{I}(B_2)$ is the resulting ideal $Stab(i, j)$. Table 10.1 shows that $Stab$ is exactly the map $\phi(i, j) = (i \cup j)0 + (i \cap j)1$ defined in Chapter 4. Note that ϕ is symmetric (i.e., $\phi(i, j) = \phi(j, i)$), so ϕ can be completely characterized by the 6

entries in the upper triangular portion of Table 10.1. There are also 6 vertices in $\mathcal{J}(B_2)$, one for each of the 6 upper triangular entries. Thus there is a 1-1 correspondence between the number of vertices in $\mathcal{J}(B_2)$ and the definition of $\phi : \mathcal{J}(B_1) \times \mathcal{J}(B_1) \rightarrow \mathcal{J}(B_2)$ in Table 10.1. This 1-1 correspondence is shown on the vertices of $\mathcal{J}(B_2)$ in Figure 10.2.

Table 10.1. Definition of $\phi = (i \cup j)0 + (i \cap j)1 : \mathcal{J}(B_1) \times \mathcal{J}(B_1) \rightarrow \mathcal{J}(B_2)$

$\mathcal{J}(B_1)$ Ideal		$\emptyset$	$\{0\}$	$\{0, 1\}$
	Maximum Element	$\emptyset$	0	1
$\emptyset$	$\emptyset$	$\emptyset$	$\{00\}$	$\{00, 10\}$
$\{0\}$	0	$\{00\}$	$\{00, 01\}$	$\{00, 01, 10\}$
$\{0, 1\}$	1	$\{00, 10\}$	$\{00, 01, 10\}$	$\{00, 01, 10, 11\}$

Let us take the same approach for $Stab : \mathcal{J}(B_2) \times \mathcal{J}(B_2) \rightarrow \mathcal{J}(B_3)$ in Figure 10.3. To simplify notation, we again represent the ideals of $\mathcal{J}(B_2)$ by their maximum elements where possible. The only ideal that is not principal, $\{00, 01, 10\}$, is represented by $-$. Also note that there are no edge weights on either $\mathcal{J}(B_2) \times \mathcal{J}(B_2)$ or $\mathcal{J}(B_3)$ to show that $Stab$ is intertwining – *because it is not*. To see why, let us build the map $Stab$ incrementally. (For reasons that will become clear, we cannot just use the equivalence classes of vertices in $\mathcal{J}(B_2) \times \mathcal{J}(B_2)$ as the vertices of $\mathcal{J}(B_3)$).

The mapping $Stab$ on ranks 0 and 1 is trivial. For rank 2, there are 3 equivalence classes of vertices in $\mathcal{J}(B_2) \times \mathcal{J}(B_2)$ from a structural viewpoint. Again, this corresponds to the number of vertices of rank 2 in $\mathcal{J}(B_3)$. A similar correspondence exists between the 3 classes of structurally equivalent vertices of rank 3 in $\mathcal{J}(B_2) \times \mathcal{J}(B_2)$ and the 3 vertices of rank 3 in $\mathcal{J}(B_3)$. The edges between ranks are determined from the pre-images.

However, the situation is less clear in rank 4. There are 5 classes of structurally equivalent vertices (although 3 of the classes look similar to each other):

$$\{\{11, \emptyset\}, \{\emptyset, 11\}\}$$

$$\{\{10, 10\}\}$$

$$\{\{01, 01\}\}$$

$$\{\{10, 01\}, \{01, 10\}\}$$

$$\{\{-, 00\}, \{00, -\}\}$$

To resolve this dilemma, observe that each vertex of rank 4 in $\mathcal{J}(B_3)$ has exactly one predecessor, with one exception. Looking at $\mathcal{J}(B_2) \times \mathcal{J}(B_2)$ in Figure 10.3, we see that each of the first three equivalence classes listed above has one equivalence class predecessor. By contrast, each vertex in either of the last two equivalence classes listed above has at least one predecessor from two different equivalence classes. Thus, under *Stab*, $\{\{ \{10, 01\}, \{01, 10\} \}, \{ \{-, 00\}, \{00, -\} \} \}$ constitutes one equivalence class of heterogeneous structure. This observation allows us to complete the definition of *Stab*.

As a cross-check, Table 10.2 gives the definition of $\phi : \mathcal{J}(B_2) \times \mathcal{J}(B_2) \rightarrow \mathcal{J}(B_3)$. Once again, the definition of ϕ matches that for *Stab* in Figure 10.3. It is important to note that the upper triangular matrix of Table 10.2 has 21 entries, whereas there are 20 vertices of $\mathcal{J}(B_3)$. Therefore, two of the upper triangular entries of Table 10.2 must be mapped to the same element by ϕ , and there is no obvious relationship between the two argument pairs. This is shown by “splitting” the pre-image arguments inside the corresponding vertex of $\mathcal{J}(B_3)$.

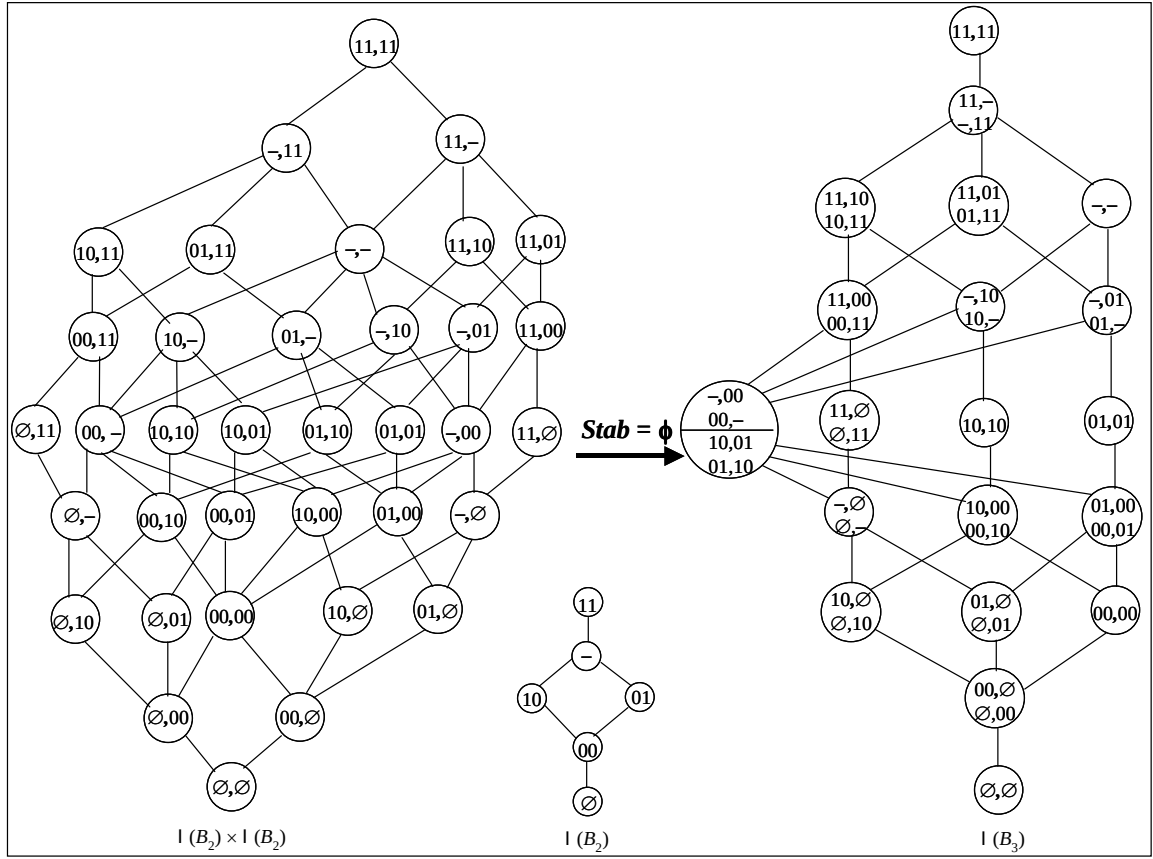


Figure 10.3. $Stab : \mathcal{I}(B_2) \times \mathcal{I}(B_2) \rightarrow \mathcal{I}(B_3)$ is not an Intertwining Operator

Now here is the key point. Each vertex in the subclass $\{\{-,00\}, \{00,-\}\}$ has a successor in all three equivalence classes of rank 5. However, each vertex in the subclass $\{\{10,01\}, \{01,10\}\}$ has only 2 successors in rank 5. Neither vertex has a successor in the class $\{\{11,00\}, \{00,11\}\}$, so there is no nontrivial assignment of weights such that $\rho(\{10,01\}, \{11,00\}) + \rho(\{10,01\}, \{00,11\}) = \rho(\{-,00\}, \{11,00\}) + \rho(\{-,00\}, \{00,11\})$, for example. Therefore $Stab$ is not an intertwining operator from $\mathcal{I}(B_2) \times \mathcal{I}(B_2) \rightarrow \mathcal{I}(B_3)$ even though $\mathcal{I}(B_3)$ is normal (and therefore RUSS). Thus, to prove that $\mathcal{I}(B_k)$ is RUSS in general (as this author conjectures); some other approach must be used.

Table 10.2. Definition of $\phi = (i \cup j)0 + (i \cap j)1 : \mathcal{J}(B_2) \times \mathcal{J}(B_2) \rightarrow \mathcal{J}(B_3)$

$\mathcal{J}(B_1)$ Ideal		$\emptyset$	00	00 10	00 01	00 10 01	00 10 01 11
	Maximum Element	$\emptyset$	00	10	01	–	11
$\emptyset$	$\emptyset$	$\emptyset$	000	000 100	000 010	000 100 010	000 100 010 110
00	00	000	000 001	000 100 001	000 010 001	000 100 010 001	000 100 010 110 001
00 10	10	000 100	000 100 001	000 100 001 101	000 100 010 001	000 100 010 001 101	000 100 010 110 001 101
00 01	01	000 010	000 010 001	000 100 010 001	000 010 001 011	000 100 010 001 011	000 100 010 110 001 011
00 10 01	–	000 100 010	000 100 010 001	000 100 010 001 101	000 100 010 001 011	000 100 010 001 101 011	000 100 010 110 001 101 011
00 10 01 11	11	000 100 010 110	000 100 010 110 001	000 100 010 110 001 101	000 100 010 110 001 011	000 100 010 110 001 101 011	000 100 010 110 001 101 011 111

Another approach to showing strong Spernerity of $\mathcal{J}(B_k)$ is to prove the existence of a symmetric chain decomposition for $\mathcal{J}(B_k)$ in general. Is there a well-defined category of symmetric chain decompositions with morphisms? If so, what are the morphisms, and does this category have pushouts? Do the non-self-dual elements of $\mathcal{J}(B_k)$ fall into complete isomorphism classes in general, as is the case for $\mathcal{J}(B_4) / S_4$? Does a symmetric chain decomposition on $\mathcal{J}(B_k) / S_k$ suffice to show that $\mathcal{J}(B_k)$ is strongly Sperner? Again, this could lead to an algorithmic approach to determine whether or not $\mathcal{J}(B_k)$ has a symmetric chain decomposition in general.

The questions in this study can also be raised for other posets. Recall that $L_{m,n}$, the set of all monotone-decreasing n -tuples of integers in $\{1, \dots, m\}$, is RUSS. What Sperner properties does $L_{m,n,p}$ have?

Like $\mathcal{J}(B_k)$, other posets of interest are geometric in nature. A *permutohedron* is an $(n - 1)$ -dimensional polytope embedded in n -space. The vertices are permutations of the coordinates of the vector $\langle 1, 2, \dots, n \rangle$. Two vertices are adjacent iff one can be obtained from the other by swapping two coordinates that differ by 1. What are the Sperner properties of permutohedra?

Another class of geometric posets are the Bruhat orders of finite Coxeter groups. A Coxeter group W is a subgroup of orthogonal transformations that are generated by reflections. There is a basis set of reflections. If W is finite, each element (reflection or rotation) g of W can be represented as the composition of basis reflections. The minimum number of such reflections is $l(g)$. The Bruhat order on W is the transitive closure of $\propto$, where $h \propto g$ if $h = Rg$ for some reflection R and $l(h) < l(g)$. The quotient of

the Bruhat order by a parabolic subgroup is a Bruhat poset. [Coxeter 1963] is the classic reference on Coxeter groups. [Harper 2004] provides a condensed treatment with emphasis on Bruhat orders and Bruhat posets. What are their Sperner properties?

BIBLIOGRAPHY

- [Awodey] Awodey, Steve, *Categories for Everybody*, to be published.
- [Birkhoff 1967] Birkhoff, Garrett, *Lattice Theory*, Third Edition, American Mathematical Society, 1967.
- [Coxeter 1963] Coxeter, H. S. M., *Regular Polytopes*, Macmillan, New York, 1963.
- [Dedekind 1897] Dedekind, J. W. R., Ueber Zerlegungen von Zahlen durch ihre gröszen gemeinsamen Teiler, *Festschrift Hoch. Braunschweig u. ges. Werke II* 1897, pp. 103-148.
- [Dilworth 1950] Dilworth, Robert P., A Decomposition Theorem for Partially Ordered Sets, *Annals of Mathematics* 51: pp. 161-166, 1950.
- [Engel 1997] Engel, Konrad, *Sperner Theory*, Cambridge University Press, Cambridge, United Kingdom, 1997.
- [Even 1979] Even, Shimon, *Graph Algorithms*, Computer Science Press, 1979.
- [Ford and Fulkerson 1962] Ford, L. R. and Fulkerson, D. R., *Flows in Networks*, Van Nostrand, Princeton, 1962.
- [Gansner] Gansner, E. R., On the Lattice of Ideals of an Up-Down Poset, preprint.
- [Goguen 1991] Goguen, Joseph A. A Categorical Manifesto, *Mathematical Structures in Computer Science*, Volume 1, Number 1, March 1991, pp. 49-67.
- [Graham and Harper 1969] Graham, R., L., and Harper, L. H., Some Results on Matchings in Bipartite Graphs, *SIAM J. Appl. Math.*, **17**, 1969, pp. 1017-1022.
- [Greene and Kleitman 1976] Greene, C., and Kleitman, D. J., Strong Versions of Sperner's Theorem, *J. Combin. Theory Ser. A*, Vol.20, 1976, pp. 80-88.
- [Griggs 1982] Griggs, Jerrold R., Collections of Subsets with the Sperner Property, *Transactions of the American Mathematical Society*, Vol. 269, No. 2, February 1982, pp. 575-591.
- [Griggs 1977] Griggs, Jerrold R., Symmetric Chain Orders, Sperner Theorems, and Loop Matchings, Ph.D. Thesis, Massachusetts Institute of Technology, 1977.
- [Hall 1998] Hall, Marshall, *Combinatorial Theory*, Second Edition, Blaisdell Publishing, Waltham, Massachusetts, 1998.

- [Hall 1935] Hall, Philip, On Representatives of Subsets, *Journal of the London Mathematical Society* 1935, s1-10: pp. 26-30.
- [Harper 2004] Harper, L. H., *Global Methods for Combinatorial Isoperimetric Problems*, Cambridge University Press, Cambridge, United Kingdom, 2004.
- [Harper 1980] Harper, L. H., The Global Theory of Flows in Networks, *Advances in Applied Mathematics* **1**, pp. 158-181, 1980.
- [Harper 1982] Harper, L. H., Morphisms for the Strong Sperner Property of Stanley and Griggs, *Proceedings of the Seminar on Combinatorics and Applications in honour of Prof. S. S. Shrikhande on his 65th birthday*, Indian Statistical Institute, December 1982, pp. 153-166.
- [Harper 1974] Harper, L. H., The Morphology of Partially Ordered Sets, *Journal of Combinatorial Theory* Vol. 17, No. 1, July 1974, pp. 44-58.
- [Harper and Bezrukov] Harper, L. H. and Bezrukov, S. L., Monte Carlo Estimation of the Number of Ideals in a Poset, to be published.
- [Kleitman 1969] Kleitman, Daniel, On Dedekind's Problem: The Number of Monotone Boolean Functions, *Proceedings of the American Mathematical Society*, Vol. 21, No. 3, pp. 677-682, June 1969.
- [Korshunov 2003] Korshunov, A. D., Monotone Boolean Functions, *Russian Mathematical Surveys* 58:5, pp. 929-1001, 2003.
- [Mac Lane 1998] Mac Lane, Saunders, *Categories for the Working Mathematician*, Second Edition, Springer-Verlag, New York, 1998.
- [Malhotra, Pramodh Kumar and Maheshwari 1978] Malhotra, V. M., Pramodh Kumar, M., and Maheshwari, S. N., An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, India, 1978.
- [Proctor 1982J] Proctor, Robert A., Representations of $sl(2, \mathbb{C})$ on Posets and the Sperner Property, *SIAM J. Alg. Disc. Meth.* Vol. 3, No. 2, June 1982, pp. 275-280.
- [Proctor 1982D] Proctor, Robert A., Solution of Two Difficult Combinatorial Problems with Linear Algebra, *American Mathematical Monthly*, Vol. 89, No. 10, December 1982, pp. 721-734.

[Proctor, Saks and Sturtevant 1980] Proctor, Robert A., Saks, Michael E, and Sturtevant, Dean G., Product Partial Orders with the Sperner Property, *Discrete Mathematics* 30 (1980), pp. 173-180.

[Rydeheard and Burstall 1988] Rydeheard, D. E., and Burstall, R. M., *Computational Category Theory*, Prentice Hall International, Hertfordshire, England, 1988.

[Saks 1979] Saks, Michael, A Short Proof of the Existence of k -saturated Partitions of Partially Ordered Sets, *Advances in Mathematics* 33, no. 3, pp. 207-211, 1979.

[Schrage and Baker 1978] Schrage, L., and Baker, K. R., Dynamic Programming Solution of Sequential Problems with Precedence Constraints, *Operations Research* 26, pp. 444-449, 1978.

[Sperner 1928] Sperner, Emanuel, Ein Satz über Untermengen einer endlichen Menge, *Mathematische Zeitschrift*, 27: pp. 544-548, 1928.

[Stanley 1991] Stanley, Richard P., Some Applications of Algebra to Combinatorics, *Discrete Applied Mathematics* 34 (1991), pp. 241-277.

[Stanley 1980] Stanley, Richard P., Weyl Groups, the Hard Lefschetz Theorem, and the Sperner Property, *SIAM J. Alg. Disc. Meth.* Vol. 1, No. 2, June 1980, pp. 168-184.

[Wiedemann 1991] Wiedemann, Doug, A Computation of the Eighth Dedekind Number, *Order* 8: 1991, pp. 5-6.

APPENDIX A: DEDEKIND NUMBERS AND RELATED SEQUENCES

The problem of counting the number of monotone Boolean functions of k variables is known as *Dedekind's problem* after J. W. R. Dedekind, who first posed the problem [Dedekind 1897]. This number is $|\mathcal{J}(B_k)|$. This problem has a rich history. [Kleitman 1969] provided lower and upper bounds to Dedekind's number. [Wiedemann 1991] computed the eighth Dedekind number. Dedekind numbers greater than 8 are unknown. [Korshunov 2003] provided two separate estimates of the Dedekind numbers: one for k odd and one for k even. [Harper and Bezrukov] have developed a Monte Carlo method for estimating the Dedekind numbers based on random walks on the ideal exchange graph.

A related sequence is the number $|\mathcal{J}(B_k) / S_k|$ of equivalence classes of ideals of the Boolean lattice. Table A.1 shows the values of these two sequences as far as is currently known. In the On-Line Encyclopedia of Integer Sequences, the integer sequence $\{|\mathcal{J}(B_k)|\}$ is A000372 and the integer sequence $\{|\mathcal{J}(B_k) / S_k|\}$ is A003182.

Table A.1. Integer Sequences for Monotone Boolean Functions

k	Dedekind's number $ \mathcal{J}(B_k) $	$ \mathcal{J}(B_k) / S_k $
0	2	2
1	3	3
2	6	5
3	20	10
4	168	30
5	7,581	210
6	7,828,354	16,353
7	2,414,682,040,998	unknown for $k > 6$
8	56,130,437,228,687,557,907,788	
9	unknown for $k > 8$	

APPENDIX B: NORMAL TEST PROGRAM

This appendix is available as a separate electronic file.

APPENDIX C: NORMAL TEST OUTPUT

This appendix is available as a separate electronic file.

APPENDIX D: INDEPENDENT VERIFICATION PROGRAM

This appendix is available as a separate electronic file.

APPENDIX E: INDEPENDENT VERIFICATION PROGRAM INPUT

This appendix is available as a separate electronic file.

APPENDIX F: INDEPENDENT VERIFICATION PROGRAM OUTPUT

This appendix is available as a separate electronic file.