DEDEKIND NUMBERS
AND RELATED SEQUENCES

by

Timothy James Yusun
B.Sc. (Hons.), Ateneo de Manila University, 2008

A Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Fall 2011

APPROVAL

Name:	Timothy James Yusun
Degree:	Master of Science
Title of Thesis:	Dedekind Numbers and Related Sequences
Examining Committee:	**Dr. Abraham Punnen**, Chair Professor
	Dr. Tamon Stephen, Senior Supervisor Assistant Professor
	Dr. Zhaosong Lu, Supervisor Assistant Professor
	Dr. Michael Monagan, External Examiner Professor, Department of Mathematics Simon Fraser University
Date Approved:	December 6, 2011

Partial Copyright Licence

Abstract

This thesis considers Dedekind numbers, where the nth Dedekind number $D(n)$ is defined as the number of monotone Boolean functions (MBFs) on n variables, and $R(n)$, which counts non-equivalent MBFs. The values of $D(n)$ and $R(n)$ are known for up to $n = 8$ and $n = 6$, respectively; in this thesis we propose a strategy to compute $R(7)$ by considering profile vectors of monotone Boolean functions. The profile of an MBF is a vector that encodes the number of minimal terms it has with $1, 2, \ldots, n$ elements. The computation is accomplished by first generating all profile vectors of 7-variable MBFs, and then counting the functions that satisfy each one by building up from scratch and taking disjunctions one by one. As a consequence of this result, we will be able to extend some sequences on the *Online Encyclopedia of Integer Sequences*, notably $R(n)$ up to $n = 7$, using only modest computational resources.

Acknowledgments

I would like to extend my gratitude to the people who have made this thesis possible:

To my supervisor Dr. Tamon Stephen, for constantly pushing me to better myself. It is with no exaggeration that I say this thesis could not have been written and completed without his guidance and for this I am enormously grateful.

To Dr. Michael Monagan for all the valuable insights and comments on my work.

To all my classmates at SFU for being in the same post-graduate boat; things are much easier to do when others are doing it with you: Annie, Brad, Farzana, Krishna, Mehrnoush, Nui, Sara, Sherry, Sylvia, Tanmay, and Yong.

To Wayne Heaslip at Health and Counselling for listening and helping me clear my mind.

To everyone at Gourmet Cup in Lansdowne Mall, for understanding when I needed to disappear for a few weeks, especially my sister Jam who is the best at what she does.

To my friends back in the Philippines and everywhere else, for being my inspirations: Chris L., Chris N., Enrique, Ian, Justin, Karol, Manny, Mathew and Nica, Oliver, Tim, and most especially Garrick, wherever you are.

To my family who will always be there for me, even if I don't answer my phone.

And finally, to Poch, for sticking it out with me while I was going crazy. You light up my life; *tayong dalawa*.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

> The hostility towards lattice theory began when Dedekind published the two fundamental papers that brought the theory to life well over one hundred years ago. Kronecker in one of his letters accused Dedekind of "losing his mind in abstractions," or something to that effect.
> For some years I did not come back to lattice theory. In 1963, when I taught my first course in combinatorics, I was amazed to find that lattice theory fit combinatorics like a shoe.
>
> ---
>
> *Gian-Carlo Rota*, The Many Lives of Lattice Theory
> *Notices of the AMS, Vol. 44, No. 11, 1997*

The end of the 19th century saw the two papers written by German mathematician Dedekind, which are considered as the cornerstones on which the theory of lattices was founded: the *Über Zerlegungen von Zahlen durch ihre größten gemeinsamen Teiler* and the *Über die von drei Moduln erzeugte Dualgruppe*. The subject was left dormant for a few decades, until in the 1930-1940's when Birkhoff published a series of papers on Universal Algebra, rediscovering Dedekind's work, and eventually writing his famous *Lattice Theory* treatise. In later years lattices have proven to be versatile in its translatability to varying fields of mathematics both pure and applied. They can be found in digital image filtering [26], universal algebra [3], formal concept analysis [14], and even counterterrorism [11].

This thesis is not about lattice theory *per se*, but rather about a number of curiosities

surrounding the Dedekind numbers and some related sequences, all of which have respective formulations in the language of distributive lattices, but which are more commonly stated using set theory, boolean logic, and combinatorial optimization. The Dedekind numbers, as we will prove in a moment, describe the number of monotone Boolean functions on n variables, the number of labeled Sperner hypergraphs on n vertices, and the number of antichains in the partially-ordered set $2^{[n]}$. Only its values up to $n = 8$ are currently known.

Furthermore, we will be looking at the set of nonequivalent monotone Boolean functions, where equivalent monotone Boolean functions can be obtained from one another via a renaming of variables. This forms an equivalence class partition of all such functions. Counting how many of these classes exist for a number n is an interesting question as well, and this is known only up to $n = 6$. We deal with this towards the end of Chapter 2 and in Chapter 3.

Monotone Boolean functions can also be classified using their minimal terms, that is, those clauses that evaluate to 1, but whose subclauses all evaluate to 0. There are a few known equations that will generate the number of functions with k terms, but their description becomes impractical for $k > 10$; see Chapter 1.

In Chapter 2 we discuss known algorithms that have been successfully used to count these numbers. We are also in the process of computing the 7th Dedekind number for nonequivalent monotone Boolean functions, using the techniques in Chapter 3. Our approach is novel in that we consider classifying MBFs according to their profile vectors, where the profile of a function is a vector that gives the number of minimal terms it has with $1, 2, 3, \ldots, n$ elements. Our strategy is to generate all the possible profile vectors of MBFs on 7 variables, and then count each one separately in succession, building up from scratch. As a result of this, we are able to introduce new sequences to the *Online Encyclopedia of Integer Sequences*, that count the number of nonequivalent n-variable monotone Boolean functions with k minimal terms, for up to $n = 7$ [27].

1.1 Setting Up

A *boolean function on n variables* (BF) is a function $f : \{0,1\}^n \to \{0,1\}$, where the values 0 and 1 are often denoted by FALSE and TRUE, respectively. A *monotone boolean function* (MBF) additionally satisfies the condition $x \leq y \Rightarrow f(x) \leq f(y)$, for any $x, y \in \{0,1\}^n$. We say that $x \leq y$ if $x_i \leq y_i$ for all $i = 1, 2, \ldots, n$. As an example, the conjunction $f(x_1, x_2) = x_1 \wedge x_2$ is monotone, taking the value 1 if $(x_1, x_2) = (1,1)$ and 0 otherwise. On the other hand, the exclusive-or function $f(x_1, x_2) = x_1 \oplus x_2$, which is true if and only if exactly one of x_1 and x_2 is true, is not monotone, since $f(0,1) = 1$ and $f(1,1) = 0$. Indeed, a BF is monotone if and only if it can be written as a combination of conjunctions and disjunctions only (while the exclusive-or function cannot be written without a negation, for instance $x_1 \oplus x_2 = (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$ is one representation).

The set $\{0,1\}^n$ is just the n-dimensional hypercube, denoted by B^n, while we define B^n_k to be the collection of vectors in B^n which have exactly k ones. We say that the sets in B^n_k are in the k-th level, or are of *rank* k. There are $\binom{n}{k}$ such sets/vectors. Another way of representing B^n is to refer to each vector by the positions in which they have a 1. So $(1,1,0,0,1)$ will be 125, $(1,0,0,1,0,1)$ is 146, etc.

Since each input state in B^n has two possible output states, there are a total of 2^{2^n} boolean functions on n variables. On the other hand, no exact closed form is known for the number of *monotone* boolean functions on n variables. This number is usually denoted by $D(n)$, which is also called the nth *Dedekind number*. Finding $D(n)$ for any n is known as *Dedekind's Problem*. The first few values are given in Table 1.1, taken from [27].

Currently, only values of $D(n)$ up to $n = 8$ are known. Kisielewicz gives in [18] a logical summation formula for $D(n)$, however it does not offer any added value as performing the computation using his summation has the same complexity as simply brute force enumeration of $D(n)$. (See [21], e.g.)

There are some asymptotic results concerning the behavior of $D(n)$, one of the earliest of which was a result of Kleitman in 1969, that $\log_2 D(n) \sim \binom{n}{\lfloor n/2 \rfloor}$ [20]. So far, the most accurate one is given by Korshunov in [21], given in Table 1.1.

n	$D(n)$	Source
0	2	Dedekind, 1897
1	3	
2	6	
3	20	
4	168	
5	7 581	Church, 1940 [5]
6	7 828 354	Ward, 1946 [28]
7	2 414 682 040 998	Church, 1965 [6] (verified by Berman and Kohler, 1976 [2])
8	56 130 437 228 687 557 907 788	Wiedemann, 1991 [30]

Table 1.1: Known Values of $D(n)$, `A000372`

$D(n) \sim 2^{\binom{n}{n/2}} \cdot \exp\left[\binom{n}{\frac{n}{2}-1}\left(2^{-n/2}+n^2 2^{-n-5}-n2^{-n-4}\right)\right]$, for even n
$D(n) \sim 2^{\binom{n}{(n-1)/2}+1} \cdot \exp\left[\binom{n}{\frac{n-3}{2}}\left(2^{(-n-3)/2}-n^2 2^{-n-5}-n2^{-n-3}\right) + \binom{n}{\frac{n-1}{2}}\left(2^{(-n-1)/2}-n^2 2^{-n-4}\right)\right]$, for odd n

Table 1.2: Korshunov's Asymptotics

Define a *minimal term* of an MBF f to be an input $x \in \{0,1\}^n$ such that $f(x) = 1$ and $f(y) = 0$ if $y < x$. The minimal terms of a monotone Boolean function represent the "smallest" sets where the function equals one – any input smaller than x evaluates to zero, and everything above evaluates to one by virtue of monotonicity. For example, the function $f(x_1,x_2,x_3) = x_1 \vee x_2 x_3$ evaluates to one at $(1,0,0)$ and $(0,1,1)$, as well as at all vectors larger than $(1,0,0)$ and $(0,1,1)$, and evaluates to 0 at all other vectors. Indeed, each MBF can be written as a disjunction of clauses, each representing one of its minimal terms. The MBF with minimal terms $(0,0,1,0)$, $(1,1,0,0)$, and $(1,0,0,1)$, as another example, is just $f(x_1,x_2,x_3,x_4) = x_3 \vee x_1 x_2 \vee x_1 x_4$. For brevity we just write $x_i \wedge x_j$ as $x_i x_j$.

MBFs can be classified according to the number of minimal terms they have. Call $D_k(n)$ the number of monotone Boolean functions on n variables with k minimal terms. Kilibarda et al. have derived closed formulas to compute $D_k(n)$ for fixed $k = 4,5,\ldots,10$, and these sequences are in the Online Encyclopedia for Integer Sequences as `A051112` to `A051118` ([17],[27]).

We define an MBF f to be *equivalent* to another MBF g if g can be obtained from f by a renaming of the variables. For example, the function $f(x_1,x_2,x_3) = x_1x_2 \vee x_2x_3$ is equivalent to $g(x_1,x_2,x_3) = x_1x_2 \vee x_1x_3$, since the permutation $\left(\begin{smallmatrix}1&2&3\\2&1&3\end{smallmatrix}\right)$ acting on the variables x_1,x_2,x_3 sends f to g. We write this as $f \sim g$.

We can prove that the notion of equivalence as defined above is an equivalence relation on $D(n)$, and hence forms an equivalence class partition of $D(n)$.

Theorem 1. *The relation "$\sim$" is an equivalence relation.*

Proof. Clearly, f is equivalent to itself via the identity permutation, so equivalence is reflexive.

Given $f \sim g$, if π is the permutation on the variables that sends f to g, then π^{-1} sends g to f. Hence equivalence is symmetric.

Moreover, if $f \sim g$ and $g \sim h$ for some MBFs f,g,h, and if π and τ are the respective permutations on the variable names, then the permutation $\pi \cdot \tau$, that is, the permutation obtained by first applying π and then τ to the variable names, will send f to h, hence $f \sim h$ and equivalence is transitive. Therefore, "$\sim$" is an equivalence relation on $D(n)$. □

Let $R(n)$ be the number of equivalence classes defined by "$\sim$" among monotone Boolean functions on n variables. As with $D(n)$, no closed form is known to compute $R(n)$, and in fact only the values up to $n = 6$ are known, as we show in Table 1.3. The numbers $R_k(n)$ are defined in a similar manner.

Example 2. The five functions in $R(2)$ are: $f = 0$, $f = 1$, $f = x_1$, $f = x_1 \vee x2$, and $f = x_1x_2$. The functions in $D(2)$ are exactly these functions plus $f = x_2$, which is equivalent to $f = x_1$.

Clearly, we have $R(n) \geq \dfrac{D(n)}{n!}$ since each equivalence class has at most $n!$ elements.

n	$R(n)$
0	2
1	3
2	5
3	10
4	30
5	210
6	16 353

Table 1.3: Known Values of $R(n)$, `A003182`

Asymptotically, we expect $R(n) \sim \frac{D(n)}{n!}$ since for large values of n, most monotone Boolean functions will have $n!$ equivalent MBFs. In fact, $D(n)$ and $R(n)$ grow so fast that the factor of $n!$ disappears into the lower-order terms in Korshunov's asymptotics.

Note that $R(7) \sim \frac{D(7)}{7!} = 4.79 \times 10^8$.

1.2 Dedekind and Some History

The German mathematician Dedekind introduced the Dedekind numbers in his *Über Zerlegungen von Zahlen durch ihre größten gemeinsamen Teiler*, published in 1897 [8]. This roughly transaltes to *About breakdowns of numbers by their greatest common divisor*, and describes the lattice of divisors of an integer with the operations gcd and lcm, and the partial order given by divisibility. A student of Gauss and close friend of Dirichlet, Dedekind made very important contributions to the fields of abstract algebra, algebraic number theory, and the foundations of real numbers and mathematics itself [23].

It took forty-three years after Dedekind's paper to compute the fifth Dedekind number. Church performed the calculation, and detailed his strategy in his paper entitled *Numerical analysis of certain free distributive structures.* The computation was done by counting nonequivalent monotone boolean functions, and then multiplying by the number of equivalence classes, for functions of the same rank. (This will be elaborated upon in the next section.) Church successfully computed $D(5) = 7581$, which as a consequence disproved a

conjecture of Birkhoff's that $D(n)$ would always be divisible by $(2n-1)(2n-2)$. After this, Ward computed $D(6) = 7828354$ in 1946, which he announced as a note posted in the *Bulletin of the American Mathematical Society*. He does not describe the method he used, but he verified Church's earlier result for $n = 5$ as well.

It was the same Church who successfully computed $D(7) = 2414682040998$ first, while it was independently computed by Berman and Köhler in 1976. $D(8)$ was in turn first computed by Wiedemann in 1991, using a similar tactic as Church [30]. It was then verified by Fidytek, et al. [12]. It is interesting to note here the difference between *enumerating* and *generating* these functions: in all of the papers mentioned, some properties of monotone Boolean functions were used, together with an underlying symmetry, to obtain the correct values for each n by performing a counting argument. These papers simply have the total number, and do not describe the functions themselves. A common strategy is to count $D(n)$ by starting with a list of functions in $D(n-2)$, and this is indeed what Wiedemann and Church do in their papers, while Fidytek et al. use $D(n-4)$ in theirs to count $D(8)$ [12]. To count $D(9)$, then, a list of the functions in $D(7)$ would be useful - this was generated by Shmulevich et al. in [26] in 1995, although $D(9)$ has not been counted yet. $D(8)$ on the other hand is too large to even try to generate.

In recent years, there has been a surge of literature concerning the applications of monotone Boolean functions. Numerous papers in biology, digital image processing, and computing science underscore the importance of knowing how these functions behave.

There is also interest in computing the duals of monotone Boolean functions. The *dual* of an MBF $f(x_1, x_2, \ldots, x_n)$ is defined as $f^d = \overline{f}(\overline{x_1}, \overline{x_2}, \cdots, \overline{x_n})$, where the bar on top of a variable signifies taking its negation. One reason why this is interesting is because computing the dual of an MBF is identical to computing the transversal hypergraph (or the blocking clutter) of the hypergraph defined by the MBF. This problem arises in many fields in combinatorics, more specifically in combinatorial optimization, and also in game theory and artificial intelligence [9]. Up to now the polynomial-time tractability of the decision version of the problem of monotone dualization has not yet been determined, though Fredman and

Khachiyan have proven that it is solvable in quasipolynomial time, i.e. in time $O(N^{\text{polylog(N)}})$, where N is the total number of clauses in the dual pair of functions [13].

1.3 Dedekind Numbers in Other Forms

1.3.1 Sperner Theory and Lattices

As we have mentioned, the Dedekind numbers are ubiquitous in the current literature. Aside from describing the number of monotone Boolean functions in n variables, $D(n)$ is also, for instance, the number of antichains in the poset $2^{[n]}$, or the power set of $\{1,2,\ldots,n\}$ partially ordered by inclusion. In order to see this, note that MBFs first of all can be formulated as being a function f from $2^{[n]}$ to $\{0,1\}$, where an input state x from $B^n = \{0,1\}^n$ corresponds to the subset X of $2^{[n]}$ containing the indices where the vector x is equal to one. In addition, a set X is said to be a *minimal term* of an MBF f if $f(X) = 1$ and $f(Y) = 0$ whenever X contains Y. The function $f(x_1,x_2,x_3) = x_1x_2 \vee x_3$ is an MBF with minimal terms $\{1,2\}$ and $\{3\}$. These minimal terms are the smallest sets (with respect to inclusion) which evaluate to 1, and because of monotonicity, the set of minimal terms of an MBF completely determines the function on all input sets - everything above evaluates to 1, and everything else evaluates to 0.

By definition, if two sets are minimal terms of an MBF f, then one cannot be contained in the other. Thus, the collection of minimal terms of any MBF must be an antichain, that is, a family of pairwise-incomparable elements of a poset. To illustrate, we list all the antichains of the posets $2^{[2]}$ and $2^{[3]}$ in Table 1.4.

$n=2$	{}, {{}}, {{1}}, {{2}}, {{1},{2}}, {{1,2}}
$n=3$	{}, {{}}, {{1}}, {{2}}, {{3}}, {{1},{2}}, {{1},{3}}, {{2},{3}}, {{1},{2},{3}}, {{1,2}}, {{1,3}}, {{2,3}}, {{1,2},{1,3}}, {{1,2},{2,3}}, {{1,3},{2,3}}, {{1,2},{1,3},{2,3}}, {{1},{2,3}}, {{2},{1,3}}, {{3},{1,2}}, {{1,2,3}}

Table 1.4: Antichains on the power sets of $\{1,2\}$ and $\{1,2,3\}$, from [29]

Antichains are also known as *Sperner families*, named in honor of Emanuel Sperner who proved the famous theorem that also bears his name. This theorem states that the size of an antichain in $2^{[n]}$ must be no larger than $\binom{n}{\lfloor n/2 \rfloor}$. There are many proofs of this fact; here we give a simple proof due to Lubell [10].

Theorem 3 (Sperner's Theorem)**.** *Any antichain on the n-set has at most $\binom{n}{\lfloor n/2 \rfloor}$ elements.*

Proof. Let $\mathcal{A}$ be an antichain on $[n] = \{1, 2, \ldots, n\}$, and let S_n be the set of all permutations on the same set. We will count the number of pairs N of (X, π) such that $X \in \mathcal{A}$, $\pi \in S_n$, and $X = \{\pi(1), \pi(2), \ldots, \pi(|X|)\}$ in two different ways. First, fix $X \in \mathcal{A}$. Then a corresponding permutation π must map $\{1, 2, \ldots, |X|\}$ bijectively onto X and $\{|X|+1, |X|+2, \ldots, n\}$ bijectively onto $[n] - X$. There are $|X|!(n-|X|)!$ such permutations, and so

$$N = \sum_{X \in \mathcal{A}} |X|!(n-|X|)!.$$

Now fix $\pi \in S_n$ instead, and assume that we have at least two corresponding sets $X, Y \in \mathcal{A}$. Without loss of generality, let $|X| \leq |Y|$. Then X and Y must have the same elements for the first $|X|$ terms, i.e. $\{\pi(1), \pi(2), \ldots, \pi(|X|)\}$ is contained in both X and Y. But this set is just X, so we have $X \subseteq Y$, which cannot happen as $\mathcal{A}$ is an antichain. Hence we conclude that for every fixed π, there is at most one corresponding set $X \in \mathcal{A}$. This implies

$$N \leq \sum_{\pi \in S_n} 1 = n!.$$

Combining the two relations, we get

$$\begin{aligned} &\sum_{X \in \mathcal{A}} |X|!(n-|X|)! \leq n! \\ &\qquad \Rightarrow \sum_{X \in \mathcal{A}} \frac{1}{\binom{n}{|X|}} \leq 1 \quad \text{(dividing by } n!\text{)} \end{aligned}$$

Since $\binom{n}{\lfloor n/2 \rfloor} \geq \binom{n}{k}$ for all $k = 0, 1, \ldots, n$, we have

$$\sum_{X \in \mathcal{A}} \frac{1}{\binom{n}{\lfloor n/2 \rfloor}} \leq \sum_{X \in \mathcal{A}} \frac{1}{\binom{n}{|X|}} \leq 1,$$

implying that $|\mathcal{A}| \leq \binom{n}{\lfloor n/2 \rfloor}$. □

A consequence of this is that any monotone Boolean function on n variables can have at most $\binom{n}{\lfloor n/2 \rfloor}$ minimal terms. This is precisely the size of $B^n_{n/2}$, and in fact, this upper bound on the number of minimal terms a monotone Boolean function can have is attained only by functions which completely reside in the middle level (for odd n) or levels (for even n) of B^n.

Another interesting technique that is often employed is the use of *symmetric chain decompositions* — in fact the above theorem is a corollary of the fact that there exists such a decomposition in the poset $2^{[n]}$. The following theorems and proofs are taken from [10] as well:

Definition 4 (Symmetric Chains)**.** In the poset $2^{[n]}$, a *chain* C is a collection of sets $C = (X_0 \lessdot X_1 \lessdot \cdots \lessdot X_h)$ where $X \lessdot Y$ means $X < Y$ and $X < Y' \leq Y \Rightarrow Y = Y'$. The chain is a *symmetric chain* if furthermore $|X_0| + |X_h| = n$ holds.

Theorem 5. *There exists a symmetric chain partition of* $2^{[n]}$.

Proof. When $n = 1$, the statement is trivial as $2^{[1]}$ itself is a symmetric chain. Assuming the claim is true for n, we will prove that it is true for $n+1$. Let $\mathcal{C}$ be a symmetric chain partition of $2^{[n]}$. For each chain $C = (X_0 \lessdot \cdots \lessdot X_h) \in \mathcal{C}$, construct new chains

$$C' = (X_0 \lessdot \cdots \lessdot X_h \lessdot X_h \cup \{n+1\})$$
$$C'' = (X_0 \cup \{n+1\} \lessdot \cdots \lessdot X_{h-1} \cup \{n+1\}).$$

Note that C'' does not exist if $h = 0$, so we omit it in this case. We claim that all such chains C' and C'' together make up a symmetric chain partition of $2^{[n+1]}$.

First, it is easy to see that these chains are symmetric, since for C',

$$\begin{aligned} |X_0| + |X_h \cup \{n+1\}| &= |X_0| + |X_h| + |\{n+1\}| \\ &= n+1, \end{aligned}$$

while for C'',

$$\begin{aligned}|X_0 \cup \{n+1\}| + |X_{h-1} \cup \{n+1\}| &= |X_0| + |\{n+1\}| + |X_{h-1}| + |\{n+1\}| \\ &= (|X_0| + |X_{h-1}|) + 2 \\ &= (n-1) + 2 = n+1.\end{aligned}$$

In both cases, we can separate the $\{n+1\}$ from inside the cardinality operation since it is not in any of the X_j's.

To prove this is a partition, we show that each set in $2^{[n+1]}$ is located in exactly one of these chains. Let $X \in 2^{[n+1]}$ with $n+1 \notin X$. Then $X \in 2^{[n]}$, and it is located in the corresponding C' of its chain in $\mathcal{C}$. If $n+1 \in X$, then if $X - \{n+1\}$ is the largest in its chain in $\mathcal{C}$, X can be found in the corresponding C', otherwise it is in C''.

That the set X occurs in at most one chain follows from the fact that either X or $X - \{n+1\}$ (either case) occurs in at most one chain in $\mathcal{C}$. □

Each symmetric chain in $2^{[n]}$ contains exactly one element of size $\lfloor n/2 \rfloor$. Hence, each symmetric chain partition contains exactly $\binom{n}{\lfloor n/2 \rfloor}$ chains. Given an antichain $\mathcal{A}$, we know that no two elements of $\mathcal{A}$ can be in the same chain. This implies that the number of elements of $\mathcal{A}$ is at most the number of chains in a symmetric chain partition of $2^{[n]}$, or that $|\mathcal{A}| \leq \binom{n}{\lfloor n/2 \rfloor}$, which is Sperner's theorem.

1.3.2 Hypergraphs and Combinatorial Optimization

In graph theory, graphs are defined to be a pair $G = (V, E)$ where V is a finite collection of elements called *vertices*, and E is a (possibly empty) set of unordered pairs of distinct vertices of G called *edges* [4]. Hypergraphs are an extension of this definition, where instead of being a collection of unordered pairs on a fixed finite set, we take a collection $(E_1, E_2, \ldots, E_m)$ of subsets of the vertex set $V = \{x_1, x_2, \ldots, x_n\}$ such that no E_j is empty, and every x_i is contained in some edge in the collection [1]. Hypergraphs are also called *set systems*, since a hypergraph is a collection of subsets of a ground set.

A hypergraph is said to have the *Sperner property* if its edges further satisfy $E_i \not\subseteq E_j$ for all $i \neq j$, that is, no edge contains another. By taking the vertex set to be $2^{[n]} = \{1, 2, \ldots, n\}$, it can be seen that each Sperner hypergraph, being a collection of pairwise-incomparable subsets of $2^{[n]}$, is an antichain on $2^{[n]}$. The correspondence then follows from the relationship between MBFs and antichains; each hyperedge in a Sperner hypergraph is a minimal term in its corresponding monotone Boolean function.

Furthermore, computing the *transversal hypergraph*, or the *blocking clutter* of a Sperner hypergraph is equivalent to computing the *dual* of the monotone Boolean function described by the hypergraph. We shall talk about this in more detail in the next chapter.

1.3.3 Other Applications

In lattice theory, each n-variable monotone Boolean function corresponds to an element in the free distributive lattice on n generators, while in nonlinear signal processing, these functions each uniquely define a *stack filter of window-width* n [26]. In coding theory, 1-semi-distance codes of length n correspond to monotone Boolean functions as well [16]. An important issue in computational learning theory is studying the learnability of monotone Boolean functions [25].

So-called *n-player simple games* in game theory are also monotone Boolean functions, with minimal terms equivalent to the *minimal winning forms* [24].

Finally, there has been much interest in monotone Boolean functions, and specifically monotone dualization, in computational biology. One example of a model where MBFs feature are biochemical reaction networks, where each chemical state can be thought of as a vertex, and where the smallest collections of states whose activation would disable the reaction can be thought of as hyperedges ([19], [15]).

Chapter 2

A Survey of Algorithms Counting MBFs

In this chapter, we discuss some known algorithms that have been used to count monotone Boolean functions. However, before going into detail, we will describe how we are representing these functions in code, and what practicalities have to be considered when programming with them.

2.1 Data Structures and Basic Operations

2.1.1 Representation of MBFs

A *truth table* for a Boolean function is a row of zeros and ones which encodes the outputs of the function corresponding to every possible input state. To illustrate, the function on three variables $f(x_1,x_2,x_3) = x_1 \vee x_2x_3$ has minimal terms $\{\{1\},\{2,3\}\}$, and it has the following truth table:

Input variables set to 1:	x_1,x_2,x_3	x_2,x_3	x_1,x_3	x_3	x_1,x_2	x_2	x_1	none
Output states:	1	1	1	0	1	0	1	0

Table 2.1: Truth table for the function $f(x_1,x_2,x_3) = x_1 \vee x_2x_3$

Note that the input states on the top row are arranged in a reverse colexicographic (or *colex*) order on $\{0,1\}^3$, defined as $x < y$ if $x \neq y$ and $x_k < y_k$ where $k = \max\{i : x_i \neq y_i\}$. Fixing this order, we are able to represent the above in the succinct form `11101010`. In general, any Boolean function on n variables can be written as a 0-1 string of length 2^n where each entry corresponds to an input state; we use this convention throughout this paper. The choice for the ordering might seem arbitrary, but it has the nice property that the first 2^{n-1} of its entries all involve setting the variable x_n to 1, and the second half has inputs with $x_n = 0$.

The truth table form is one of the most compact ways to represent general Boolean functions. For monotone Boolean functions, it may be more compact to just use the minimal terms, but it is more practical to use the truth table form as it encodes useful data.

As another example, the functions in $D(2)$, written in truth table form are {`1111`, `1110`, `1100`, `1010`, `1000`, `0000`}. The colexicographic order for two variables is $\{1,2\} > \{2\} > \{1\} > \{\}$.

For our computations, we decide to implement the algorithms on MATLAB, a high-level technical computing language developed by MathWorks, Inc. We use MATLAB for building a prototype because it is easy to get started, and it is built for handling large vectors and matrices. It has many built-in functions that work well with the types of lists we are generating, and if desired, further work can be transported over to other programming languages. We expect that this code would be faster if translated to C.

In MATLAB, we represent MBFs as 1×2^n row vectors. Many of the built-in functions in MATLAB such as `bitor`, `bitand`, and `find`, offer a convenient way of performing operations on these MBFs. For instance, to test whether two functions f and g satisfy $f \geq g$, we can either just type $f \geq g$ to obtain the indices of entries fulfilling this condition, or use `bitand`$(f,g) == g$, which should be true if and only if $f \geq g$. Testing for equality is simple in MATLAB as well.

The truth table form furthermore lends itself well to bit compression using 32-bit integers, i.e. the `uint32` data type in MATLAB, which encodes unsigned integers in the range of $[0, 2^{32} - 1]$. To elaborate, given an MBF of length 2^n, we partition the zeros and ones into

blocks of length 32, which we consider as a binary number $(a_0a_1a_2\ldots a_{31})_2$, and which we then convert into decimal format, by computing $\sum_{k=0}^{31} a_k 2^k$. If n is smaller than 5 and hence the truth table form has fewer than 32 entries, we just assume the rest of the entries to be zeros. For this case and when $n = 5$, the result is just one number in the interval $[0, 2^{32}-1]$.

Example 6. The MBF $f(x_1,x_2,x_3,x_4,x_5) = x_1x_3x_4 \vee x_2x_3x_4 \vee x_2x_5 \vee x_4x_5$ has minimal terms $\{1,3,4\}$,$\{2,3,4\}$,$\{2,5\}$,$\{4,5\}$, and in truth table form is written as

$$f = \texttt{11111111110011001100100000000000}.$$

Conversion to a 32-bit integer gives $2^0+2^1+\cdots+2^{10}+2^{13}+2^{14}+2^{17}+2^{18}+2^{21}$, or 1258495.

Example 7. The six-variable MBF

$$f = \texttt{11111110111111101111110010000000111110101110101011111000000000000}$$

has 64 entries, so it is divided into two blocks of 32:

$$\texttt{11111110111111101111110010000000} \rightarrow 20938623$$
$$\texttt{11111010111010101111100000000000} \rightarrow 2053983$$

hence the 32-bit integer representation of f is $(20938623, 2053983)$. Its minimal terms are $\{1,2,4\}$, $\{3,4\}$, $\{1,5\}$, $\{2,3,5\}$, $\{1,2,3,6\}$, $\{2,4,6\}$, $\{2,5,6\}$, and $\{3,5,6\}$.

In a similar way, it is not difficult to do the reverse operation of expanding these integers back into their 0-1 truth table forms. We use both these representations in different parts of the algorithms that follow.

2.1.2 List Data Structures

The algorithms in this thesis, particularly our profile generation approach, involve building and frequently referencing a very long list of integers. Thus a good implementation of such a list is crucial.

There are at least three standard ways to approach this. The first way is just to naively use arrays and the MATLAB built-in `find`, plus its reindexing operations to insert. Lookup can be done in $O(\log n)$ time, while insertions are done in $O(n)$. The second way is to use a

balanced binary tree to lookup and insert values. A binary search tree is a tree where each node represents a value, and all nodes to its left [right] have a corresponding value that is smaller [larger] than the value at the parent node [7]. The efficiency of the operations done on such trees depend directly on the height of the tree – if the tree had just one branch, for example, then searching and insertion would take the same time as naively using an array. We can construct these trees so as to minimize their height. For this case, lookups and insertions can be done in $O(\log n)$ time [7]. The third way is to use hash tables to store and retrieve data.

Hash tables allow us to do lookups more quickly. A *hash table* is a data structure that assigns associated values called *keys* to input values that have to be sorted or looked up. A *hashing function* is used to compute the key from the input value, which in this case are the MBFs, and this will be the index in the output array of the code. What this accomplishes is that whenever the output needs to be checked whether or not it contains a given MBF, we can just evaluate the hashing function on the MBF to get its key – no need to search the list.

There are some issues here, depending on the hashing function used. For example, there could be a significant number of functions who share a key with another MBF. This is called a *collision* in the computing science literature; a good hash function has very few collisions on the input set. There is a way to work around this issue, by simply moving down one row in the output list whenever a key is encountered again, but we still want to reduce the number of collisions as this will reduce the time it will take to look an MBF up in the list. Note that in the extreme case, the hash function which maps all inputs to one key would just be a regular listing of all these functions, and lookups will take linear time. Typically, hash table lookups and insertions take a constant time $O(1)$ [7].

Hashing 5-variable Boolean functions (which only have one component in their 32-bit representation) can be done by performing modular operations on the input, where the divisor is some large prime number. To make sure that few collisions occur, we use a divisor which is about double the total number items that we will generate in the list.

To hash 6- and 7-variable functions, we need additional operations in order to generate keys. One way to do this is to start by taking the exclusive-ors of their component bits to get a 32-bit integer. It may be helpful to shift or reverse the order of the bits to make the code more robust.

For instance, given the function $f = (218623, 71167, 196863, 65791)$, if we take the exclusive-ors of all four components, we obtain the key 16384. If we reverse the bits of the first and the fourth entries before taking exclusive-ors, we obtain the number 4278282922. Choosing a random large prime, say 15485867, we perform modular division and we obtain the key 4183630.

Because each of the four components are related to one another, it is much better to do a shift or a reversing operation before taking exclusive-ors.

Another hash function that is effective in practice is the polynomial hash function, which acts on the four integers (say b_1, b_2, b_3, and b_4) by repeatedly adding a number $\alpha > 2$ modulo a prime p, and multiplying by the next component. This can be written as $b_1(\alpha + b_2(\alpha + b_3(\alpha + b_4)))$ where all operations are done modulo p.

In general, choosing which divisors to use for modular operations in a hash function is crucial because this will determine the total number of keys that can be generated – and hence the size of the lookup table that must be stored in memory while the program is running.

In our computations, using hash tables for list handling instead of binary search led to a speedup of a factor of 4 for lists of size 40000, and a factor of 8 for lists of size 1500000.

2.2 Counting $D(n)$

The following is a survey of known algorithms that compute $D(n)$.

2.2.1 Generating $D(n)$ from $D(n-1)$

This algorithm makes use of the fact that the n-hypercube is the Cartesian product of the $(n-1)$-hypercube and $\{0,1\}$. It follows almost directly from what is called the *Shannon*

decomposition of a monotone Boolean function f ([9], [12]):

Lemma 8 (Shannon decomposition)**.** *Every n-variable MBF f can be written uniquely as*

$$f(x_1,x_2,\ldots,x_{n-1},x_n) = x_n f_1(x_1,x_2,\ldots,x_{n-1}) \vee f_0(x_1,x_2,\ldots,x_{n-1})$$

where $f_1 = f(x_1,x_2,\ldots,x_{n-1},1)$, $f_0 = f(x_1,x_2,\ldots,x_{n-1},0)$, and both f_0 and f_1 are monotone.

Proof. Let f be a monotone Boolean function on n variables. Recall that any MBF can be written as a disjunction of clauses, each one of which correponds to one of its minimal terms. Let $S_1,S_2,\ldots,S_k$ be the clauses in this description of f which contain x_n, and denote by $T_1,T_2,\ldots,T_l$ the rest of the clauses. We therefore have

$$\begin{aligned} f(x_1,x_2,\ldots,x_n) &= (S_1 \vee S_2 \vee \cdots \vee S_k) \vee (T_1 \vee \cdots \vee T_l) \\ &= x_n \wedge (S_1' \vee S_2' \vee \cdots \vee S_k') \vee (T_1 \vee \cdots \vee T_l) \end{aligned}$$

where each S_i' is just $S_i - x_n$. When $x_n = 1$, we have $f_1 = f(x_1,\ldots,x_{n-1},1) = (S_1' \vee S_2' \vee \cdots \vee S_k') \vee (T_1 \vee \cdots \vee T_l)$, and when $x_n = 0$, we have $f_0 = f(x_1,\ldots,x_{n-1},0) = (T_1 \vee \cdots \vee T_l)$. Hence f_0 is the function whose clauses are exactly all clauses in f that do not contain x_n, while f_1 is the function whose clauses are exactly all the clauses in f minus x_n.

One implication of this is that f_0 and f_1 are monotone as well, since they are just disjunctions of clauses, and they don't involve negations. Furthermore, f_0 and f_1 are uniquely determined – they are exactly the set of clauses described above. □

Lemma 9. *The functions f_1 and f_0 in the Shannon decomposition of f satisfy $f_0 \leq f_1$.*

Proof. For any fixed $x_1,x_2,\ldots,x_{n-1}$, we must also have $f(x_1,\ldots,x_{n-1},0) \leq f(x_1,\ldots,x_{n-1},1)$ by the monotonicity assumption on f. Hence, $f_0 \leq f_1$ for any values of the first $n-1$ variables, implying $f_0 \leq f_1$. □

We can reverse the process by starting with two MBFs from $D(n-1)$, say f and g such that $f \leq g$. Then we obtain the function $F = x_n g(x_1,\ldots,x_{n-1}) \vee f(x_1,\ldots,x_{n-1})$, which is in $D(n)$. It becomes even easier if we use the binary 2^n-tuple convention, since F can be obtained by concatenating g and f (in that order).

Example 10. Let f be the function `1000` and g the function `1110`. It is easy to see that both are monotone. Also, $f \leq g$ since $f(x) \leq g(x)$ for all possible truth inputs. Thus, $F = x_n g \vee f$ is also monotone, and it is the function $F =$`11101000`.

Therefore, to count $D(n)$, it is sufficient to count the number of pairs of functions (f,g) from $D(n-1)$ that satisfies $f \leq g$. This gives us Algorithm 1.

Algorithm 1: Generating $D(n)$ from $D(n-1)$

```
Input: D(n−1) in truth value form
Output: D(n) in truth value form
initialize D(n) ;
for f ∈ D(n−1) do
    for g ∈ D(n−1) do
        if f ≤ g then
            F := concatenate(g, f);
            add F to D(n);
        end
    end
end
```

2.2.2 Enumerating $D(n)$ from $D(n-2)$

Another way to count $D(n)$ starts from $D(n-2)$. The advantage of this is that we have a smaller input to work with, however this method will not give us a list of elements from $D(n)$ but only the cardinality. Wiedemann first mentioned this algorithm in his paper [30] and Fidytek et. al. details it in theirs [12].

Recall from the previous section that the *Shannon decomposition* of a function enabled us to effectively generate $D(n)$ by counting pairs in $D(n-1)$. This second algorithm goes one step further and decomposes the child functions, working with 4-tuples in $D(n-2)$.

Lemma 11. *Every n-variable MBF f can be written as*

$$f(x_1,\ldots,x_{n-1},x_n) = x_{n-1}x_n f_{11} \vee x_{n-1} f_{10} \vee x_n f_{01} \vee f_{00}$$

where $f_{ab} = f(x_1,\ldots,x_{n-2},a,b)$, $f_{11}, f_{01}, f_{10}, f_{00}$ are monotone, $f_{00} \leq f_{01} \leq f_{11}$ and $f_{00} \leq f_{10} \leq f_{11}$. Moreover, this decomposition is unique [12].

Proof. We can apply Lemma 8 thrice to get:

$$\begin{aligned} f(x_1, x_2, \ldots, x_{n-1}, x_n) &= x_n f_1 \vee f_0 \\ &= x_n [x_{n-1} f_{11} \vee f_{01}] \vee [x_{n-1} f_{10} \vee f_{00}] \\ &= x_{n-1} x_n f_{11} \vee x_n f_{01} \vee x_{n-1} f_{10} \vee f_{00} \end{aligned}$$

By the same lemma, all of $f_{11}, f_{01}, f_{10}, f_{00}$ are monotone and this decomposition is unique. Also, from Lemma 9 we have the relations $f_{00} \leq f_{10}$ and $f_{01} \leq f_{00}$. If we decompose by x_{n-1} first instead of x_n then we will get the same result, and the relations $f_{00} \leq f_{01}$ and $f_{10} \leq f_{11}$. Combining these we get $f_{00} \leq f_{10} \leq f_{11}$ and $f_{00} \leq f_{01} \leq f_{11}$, as desired. □

Therefore, to count $D(n)$, it is sufficient to count 4-tuples of functions in $D(n-2)$ that satisfy the conditions above. We state this as a theorem and give an example.

Theorem 12. *There is a one-to-one correspondence between $D(n)$, the set of n-variable MBFs, and the set $\left\{ (f_{11}, f_{01}, f_{10}, f_{00}) \in [D(n-2)]^4 : f_{00} \leq f_{10} \leq f_{11} \text{ and } f_{00} \leq f_{01} \leq f_{11} \right\}$ [12].*

Example 13. Let $F = \texttt{1110101011000000}$ be an MBF in $D(4)$. Note that $F(X) = 1$ when $X = \{1,2,3,4\}, \{2,3,4\}, \{1,3,4\}, \{1,2,4\}, \{1,4\}, \{1,2,3\}, \{2,3\}$. Using the previous lemma, we can decompose F into $x_3 x_4 f_{11} \vee x_4 f_{01} \vee x_3 f_{10} \vee f_{00}$, where

$$f_{00} = \texttt{0000}$$

$$f_{10} = \texttt{1100}$$

$$f_{01} = \texttt{1010}$$

$$f_{11} = \texttt{1110}$$

Clearly we have $f_{00} \leq f_{01} \leq f_{11}$ and $f_{00} \leq f_{10} \leq f_{11}$. Also, they are all monotone.

To count $D(n)$, we do not iterate through $D(n-2)$ four times; instead, we fix two functions to be f_{10} and f_{01}, and then count the number of ways of choosing MBFs from $D(n-2)$ for the other two functions. To this end, we have Lemma 14, which is also in [12].

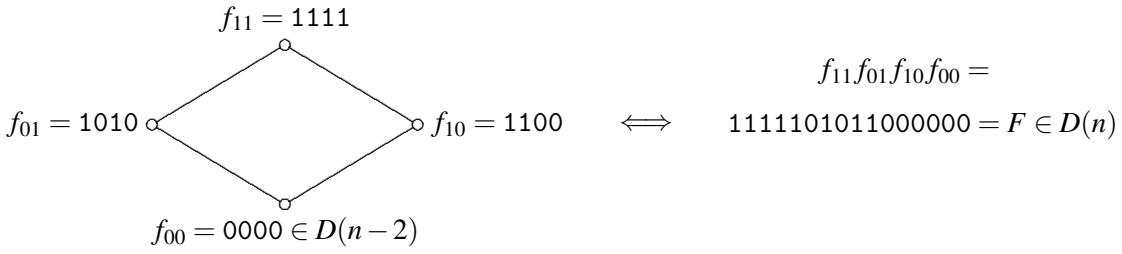

Figure 2.1: Illustration for Example 13: Two representations of an MBF in $D(4)$

Lemma 14. *Given any three MBFs f, g, and h from $D(n)$, we have*

$$h \geq f \text{ and } h \geq g \text{ if and only if } h \geq (f \vee g)$$

and

$$h \leq f \text{ and } h \leq g \text{ if and only if } h \leq (f \wedge g).$$

Proof. We will only show the first case; the second situation is dealt with via a similar argument.

The truth table for the functions f, g, and $f \vee g$ for an input value x is given in Table 2.2. We can directly infer that $h \geq (f \vee g) \iff h \geq f$ and $h \geq g$ by simply doing a case by case comparison. □

$f(x)$	$g(x)$	$f(x) \vee g(x)$
0	0	0
0	1	1
1	0	1
1	1	1

Table 2.2: Truth table for $f \vee g$

This lemma implies that instead of counting the number of functions $h \in D(n-2)$ that satisfy $h \leq f$ and $h \leq g$, it suffices to check for the condition $h \leq (f \wedge g)$. The same goes for checking if $h \geq (f \vee g)$.

We can also simplify things by using the *dual* of a BF.

Definition 15. Given a BF $f = f(x_1, x_2, \ldots, x_n)$, its *dual* is defined to be $f^d = \overline{f}(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$, where $\bar{x}$ is the negation of x.

Lemma 16. *The following are properties of a Boolean functions and their duals:*

1. *Given a formula for a BF f, its dual can be obtained by exchanging $\vee$ and $\wedge$ as well as constants 0 and 1.*
2. *Given a BF f in truth table form, its dual is obtained by taking its complement and reading the string from right to left.*
3. *If f is an MBF, then the minimal terms of its dual f^d are exactly the set of minimal transversals of the minimal terms of f.*
4. *$(f \vee g)^d = f^d \wedge g^d$ for any BFs f, g.*
5. *If $f \geq g$, then $f^d \leq g^d$.*

Proof. The first property follows from the definition of a dual MBF and de Morgan's laws, while the second property follows from the fact that the set in the ith position of the truth table form is exactly the complement of the set in the $(2^n + 1 - i)$th position, in the reverse colex order.

To see why the third property is true, let f be a monotone Boolean function whose minimal terms are the clauses $S_1, S_2, \ldots, S_k$. Then we can write f as the disjunction of all these clauses, $f = \bigvee_{i=1}^{k} S_i$. By the first property, the dual of f is thus $f^d = \bigwedge_{i=1}^{k} T_i$, where T_i contains exactly the same variables as S_i does, but taking disjunctions instead of conjunctions within each term. By distributivity of conjunctions over disjunctions, we can write f as the disjunction of all possible clauses obtained by taking one element from each T_i and taking their conjunctions.

After eliminating duplicates and terms which are already implied by another, we obtain the minimal terms of the function f^d. Two things characterize each of these clauses: first, they should contain at least one element from each clause in the original f; and secondly,

they should have no duplicates, nor be implied by another clause in f^d. In other words, each minimal term in f^d should be a minimal transversal of the minimal terms of f.

The fourth property is also a corollary of the first, in that exchanging the operations $\vee$ and $\wedge$ can be done in any order within the function formula.

Lastly, that $f \geq g$ implies $f^d \leq g^d$ follows immediately from the second property. □

Example 17. Let $f = x_2 \vee x_1x_3$. Then in truth value form, $f =$ `11101100`. Also, f is monotone, with minimal terms $\{\{2\},\{1,3\}\}$. By the first property, the dual of f is $f^d = x_2(x_1 \vee x_3) = x_1x_2 \vee x_2x_3$, also $f^d =$ `11001000` in truth value form via the second property. The minimal terms of f^d are $\{\{1,2\},\{2,3\}\}$, which are the minimal transversals of the minimal terms of f.

From the properties above, we can modify the method into counting the number of functions h which are less than $(f^d \wedge g^d)$ instead of those which are greater than $(f \vee g)$. For brevity let $\mu(f) = |\{g \in D(n) : g \leq f\}|$. (The value of n should be clear from the context.) This finally gives us Algorithm 2.

Algorithm 2: Enumerating $D(n)$ from $D(n-2)$

Input: $D(n-2)$ in truth value form
Output: $|D(n)|$
initialize `result` $= 0$;
for $f \in D(n-2)$ **do**
  for $g \in D(n-2)$ **do**
    `result` := `result` $+ \mu(f \wedge g) \times \mu(f^d \wedge g^d)$;
  end
end

Fidytek et al. also enumerated $D(8)$ using an algorithm that computes $D(n)$ from $D(n-4)$, but it is too complicated to discuss here.

2.2.3 Enumerating $D(n)$ from $D(n-2)$ and $R(n-2)$

So far, the algorithms presented have provided simple enough ways to compute the cardinality of $D(n)$ for any n. However, they are also very limited in terms of efficiency. For instance,

the two `for` loops in Algorithm 2 slow it down considerably, especially when $D(n-2)$ is large, as is the case with going from $n=6$ or $n=7$ to $n=8$ or $n=9$, respectively. In fact, computing $D(8)$ in this manner will take about 6×10^{13} iterations. Note that computing $\mu(f)$ for any $f \in D(n)$ can be done as preprocessing, and will just involve a lookup operation in the execution of Algorithm 2, though this can also take a bit of time depending on the method used. The duals can also be precomputed as well.

The algorithm we present here is in Wiedemann's paper, and he uses this to compute $D(8)$ [30]. It took him 200 hours on a Cray-2 processor to carry out the calculations.

One way to speed up implementation is to exploit the symmetries inherent in BFs - recall that two BFs are equivalent, or $f \sim g$, if one can be obtained from the other via a permutation of the variables. Denote by $R_1, R_2, \ldots, R_{R(n)}$ the equivalence classes of MBFs in $D(n)$. So for $n=6$, for instance, we would have the classes $R_1, R_2, \ldots, R_{16353}$. Also, let $h_1, h_2, \ldots, h_{R(n)}$ denote a set of fixed representatives for each class.

The key thing to notice is that $\mu(f) = \mu(g)$ if $f \sim g$. This is true because if $\pi \in S_n$ is the permutation that takes f to g, then for any other function h which is contained in f, the function obtained by applying the same permutation π on h will be less than g. This is a one-to-one correspondence, hence we have $\mu(f) = \mu(g)$.

In addition, we also have $(f \wedge g) \sim (f \wedge g)^\pi = (f^\pi \wedge g^\pi)$ for any two MBFs f, g. Thus, given any function f in the kth equivalence class, and any other function g, $\mu(f \wedge g) = \mu(h_k \wedge g^\pi)$, where π is the permutation that takes f to h_k. For the term involving duals, we only need to use the fifth property in the previous subsection to see that $(f^d \wedge g^d) = (f \vee g)^d \sim (h_k \vee g^\pi)^d = (h_k^d \wedge (g^\pi)^d)$.

These observations imply that instead of computing $\mu(f \wedge g) * \mu(f^d \wedge g^d)$ for all pairs in $[D(n-2)]^2$, we can fix f to range over all h_k's, multiplying each product by an additional term $|R_k|$ (since the value of μ will be the same). This algorithm is given as Algorithm 3.

Note that this algorithm still has to perform the hash table lookup on $D(n-2)$, but this is still a significant improvement over Algorithm 2. Moreover, we still have to generate $R(n-2)$ to start, which will be discussed in Section 2.3. After preprocessing, Algorithm 3 performs

Algorithm 3: Enumerating $D(n)$ from $D(n-2)$ and $R(n-2)$

Input: $D(n-2)$ and $R(n-2)$ in truth value form
Output: $|D(n)|$
initialize `result` ;
for $k = 1$ **to** $|R(n-2)|$ **do**
  for $g \in D(n-2)$ **do**
    `result` := `result` + $|R_k| * \mu(h_k \wedge g) * \mu(h_k^d \wedge g^d)$;
  end
end

$O(R(n-2) \cdot [D(n-2)]^2)$ operations at worst, depending on how efficiently the lookups to retrieve $\mu(f)$ are done. In contrast, Algorithm 2 performs $O([D(n-2)]^3)$ operations at worst.

We implemented Algorithms 1, 2, and 3 as a warm up, and though we were able to successfully enumerate $D(7)$ within 180 CPU seconds using Algorithm 3, it would still require a nontrivial amount of computational resources to compute $D(8)$.

2.2.4 Generating $D(n)$ via Iterative Disjunctions of Minimal Terms

The three algorithms discussed thus far made use of an available list of MBFs in a smaller number of variables to generate $D(n)$. Shmulevich, et. al. in [26] use a different way of generating $D(n)$ by starting with the list of all possible minimal terms in n variables, and then recursively taking disjunctions to build up all functions in $D(n)$.

As an illustration, for $n = 2$ there are 4 MBFs with exactly one minimal term – one for each set in $2^{[2]} = \{\{\},\{1\},\{2\},\{1,2\}\}$. Hence by the notation introduced in Chapter 1, we can write $D_1(2) = 4$. We then take disjunctions of these terms to get the members of $D_2(2)$, which as it turns out is only $x_1 \vee x_2$. Hence, we have all MBFs in two variables, shown in Table 2.3.

In general, to produce the list of functions in $D_{k+1}(n)$, we take the functions in $D_k(n)$ and form disjunctions with every function in $D_1(n)$. Note that not all functions resulting from this operation will be a function with $k+1$ minimal terms, as *subsumption* might occur. This happens when one function taken from $D_k(n)$ has a minimal term that is comparable to the

k	$D_k(2)$
0	0
1	1, x_1, x_2, x_1x_2
2	$x_1 \vee x_2$

Table 2.3: $D(2)$ classified by number of minimal terms

term taken from $D_1(n)$. For example, when going from $D_3(4)$ to $D_4(4)$, we might take the disjunction of the function $f(x_1,x_2,x_3,x_4) = x_1x_2 \vee x_2x_3 \vee x_1x_3x_4$ with $g(x_1,x_2,x_3,x_4) = x_1x_2x_3$. This will not produce a function with four minimal terms since the new term $x_1x_2x_3$ is implied by either of the first two terms in f.

To generate $D(n)$, we run the recursion step for $k = 1,2,\ldots,\binom{n}{\lfloor n/2 \rfloor}$, checking for subsumption along the way. To save space, the list $D_k(n)$ can be discarded once $D_{k+1}(n)$ has been generated, if we were only concerned about enumeration. It is easy to generate $D_1(n)$ as it is exactly $2^{[n]}$. We state the algorithm as Algorithm 4.

Algorithm 4: Generating $D(n)$ via Iterative Disjunctions

```
Output: D(n)
initialize result ;
generate D_1(n) ;
for k = 1 to (n choose ⌊n/2⌋) do
    for f ∈ D_k(n) and g ∈ D_1(n) do
        compute h = f ∨ g ;
        if no subsumption occurred then
            D_{k+1}(n) ← D_{k+1}(n) ∪ {h};
        end
    end
end
```

Shmulevich et. al. were able to use this algorithm to generate $D(7)$, with the byproduct $D_k(7)$ for all $k = 1$ to $k = 35$. However, they do not mention any attempt to enumerate $D(9)$ using their lists for $D(7)$.

2.3 Counting $R(n)$

So far, we have seen various algorithms that either generate or enumerate $D(n)$. In comparison, there have not been many efforts to compute $R(n)$. To obtain $R(n)$, one could check each function in the list against every other function for equivalence. However, as the number of permutations is $n!$ for an MBF on n variables, it becomes a challenge to be able to test for equivalence using a reasonable amount of time and memory.

This problem is known in the literature as the *Hypergraph Isomorphism Problem*. While the brute force method of testing for equivalence runs in $O(n!)$ time, there are a number of asymptotically faster algorithms, e.g. Luks proves in [22] that it is possible to test isomorphism in $O(c^n)$ time. Since we are looking to perform the computations for small values of n, the asymptotics of which algorithm we use will not be as significant.

To permute MBFs, we have to determine how the renaming of n variables translates into a permutation on the 2^n elements in the truth table form. This is simple to do in MATLAB. To do this, we take the reverse lexicographic order discussed at the start of this chapter and perform the re-indexing on its elements.

For instance, say we compute the equivalent form of the three-variable MBF $f = \texttt{10101010}$ under the permutation $\pi = \left(\begin{smallmatrix}1&2&3\\2&3&1\end{smallmatrix}\right) \in S_3$. The reverse lexicographic order and the resulting rearrangement of f can be found in Table 2.4.

Input variables of f set to 1:	x_1,x_2,x_3	x_2,x_3	x_1,x_3	x_3	x_1,x_2	x_2	x_1	none
Outputs:	1	0	1	0	1	0	1	0
For $\pi(f)$ where $\pi = (123)$:	x_1,x_2,x_3	x_1,x_3	x_1,x_2	x_1	x_2,x_3	x_3	x_2	none
Corresponding outputs:	1	1	1	1	0	0	0	0

Table 2.4: Computing a new truth table order under a permutation of the variables

If we index the $2^3 = 8$ entries of the truth table form as a vector, then performing the permutation $\left(\begin{smallmatrix}1&2&3\\2&3&1\end{smallmatrix}\right)$ on any MBF f should be equivalent to applying the permutation $\left(\begin{smallmatrix}1&2&3&4&5&6&7&8\\1&5&2&6&3&7&4&8\end{smallmatrix}\right)$ on the truth table form of f, regardless of the value of the outputs. Consequently, we can preprocess each permutation in S_n to convert it into its truth table

permutation in S_{2^n}. In this way, we are able to perform permutations by referring to this table for re-indexing.

In terms of actually deciding whether two functions are equivalent, we can achieve what the algorithm needs by generating the whole equivalence class for each function to be considered, and then taking the "least representative," that is, the lexicographically (since the 32-bit integer representation will have more than one entry) smallest MBF that is equivalent to it. In this manner we avoid having to compare functions to one another, instead just having to do this calculation immediately after each function is generated and before it is added to the final output.

Here is another example: Consider the four-variable function $f = \texttt{1111110011001000}$. To find its least representative, we generate all $4! = 24$ functions in its equivalence class, convert them to 32-bit integers, and take the smallest resulting integer. For f as given, which is represented by the number 4927, the smallest integer obtained from these calculations is 895, or the function $f^* = \texttt{11111111011000000}$. Via a routine hash lookup, we then check if f^* is already in the output matrix. If it is, then we discard it, otherwise we insert it in the correct position to preserve the sorted order of the matrix. Then we proceed to the next iteration as usual.

To enumerate $R(n)$ using any of the previously-discussed techniques, we just add a few lines of code that do the above computation to the algorithm. There might be other ways to remove duplicates that involve counting orbits of permutations, for example, but the method we use is effective for a problem of this magnitude.

Chapter 3

Profile Generation and Counting $R(7)$

3.1 Profiles of Monotone Boolean Functions

The algorithms in chapter 2 that compute $D(n)$ use $D(n-1)$, $D(n-2)$, and $R(n-2)$ to do so. Furthermore, we explained how considering $R(n-2)$ lends some symmetry, and hence makes the algorithm run faster. In this chapter, we focus on another way of classifying and generating these MBFs, and use this to enumerate $R(7)$. It is natural to refine the classification of monotone Boolean functions by number of minimal terms, and consider how many elements are contained in each of these terms. For instance, the functions `11111100` and `11111000` both have two minimal terms each, but the first function has two 1-sets as minimal terms ($\{2\},\{3\}$) while the second has a 1-set and a 2-set ($\{1,2\},\{3\}$). We define the notion of a profile formally as given in [10], and introduce some notation:

Definition 18 (Profile of an MBF)**.** Given an n-variable MBF f where $f \not\equiv 1$, the **profile** of f is a vector of length n $(a_1, a_2, \ldots, a_n)$, with the ith entry equal to the number of minimal terms of f which are i-sets.

Example 19. The MBF `11111100` has profile $(2,0,0)$ and the MBF `11111000` has profile $(1,1,0)$.

Definition 20. Given a profile vector $(a_1, a_2, \ldots, a_n)$, define $(a_1, a_2, \ldots, a_n)_D$ to be the number of monotone Boolean functions on n variables with profile vector $(a_1, a_2, \ldots, a_n)$. Similarly define $(a_1, a_2, \ldots, a_n)_R$ for nonequivalent monotone Boolean functions on n variables.

Note that the number of variables n is implicit in the profile vector - it is just the length of the vector. There are a few other properties which we discovered, and we state and prove these in Lemma 21.

Lemma 21. *Assume that all profile vectors pertain to MBFs on n variables, unless otherwise stated.*

(A) $(0,0,\ldots,a_i,\ldots,0)_D = (0,0,\ldots,\binom{n}{i} - a_i,\ldots,0)_D$.

(B) *If* $a_1 > 0$, *then* $a_n = 0$ *and* $(a_1, a_2, \ldots, a_{n-1}, a_n)_D = (a_1 - 1, a_2, \ldots, a_{n-1})_{D_{n-1}}$.

(C) $(a_1, a_2, \ldots, a_{n-2}, a_{n-1}, a_n)_D = (a_{n-1}, a_{n-2}, \ldots, a_2, a_1, a_n)_D$.

All these statements hold true when D *is replaced by* R.

Proof. The proof of each claim rests on the fact that there is a one-to-one correspondence between functions of the first type and functions of the second type, for the purposes of counting both $D(n)$ and $R(n)$.

For **(A)**, given an MBF with exactly a_i i-sets as minimal terms, we can derive another MBF with minimal terms exactly the $\binom{n}{i} - a_i$ i-sets which were not taken in the first MBF. Furthermore, the images of any two equivalent functions under this correspondence will also be equivalent, under the same permutation.

Part **(B)** follows from the Shannon decomposition of an MBF discussed in Section 1. Given a function f with a profile that satisfies $a_1 > 0$, it has a singleton set that is a minimal term, say $\{n\}$. Decomposing over the variable x_n, we get $f = x_n f_1 \vee f_0$. But $f_1 = f(x_1, x_2, \ldots, x_{n-1}, 1) = 1$ since $\{n\}$ is a minimal term, so $f = f_0 = f(x_1, x_2, \ldots, x_{n-1}, 0)$, which is an $(n-1)$-variable MBF with profile $(a_1 - 1, a_2, \ldots, a_{n-1})$. All minimal terms are retained, and just the singelton set is removed.

For **(C)**, if $a_n = 1$, then $\{1, 2, \ldots, n\}$ is the only minimal term. This implies that all the other a_i's are zero, and hence the claim follows trivially.

If $a_n = 0$, assume that the minimal terms of a given MBF f are $A_1, A_2, \ldots, A_k$. We know that none of the A_i's are comparable, so it should follow that none of the sets $[n] - A_1, [n] - A_2, \ldots, [n] - A_k$ must be comparable as well, making the collection $\{[n] - A_j\}_{1 \leq j \leq k}$ an antichain, i.e. an MBF g where the number of i-sets is equal to the number of $(n-1)$-sets in f. This proves that the profile of g is $(a_{n-1}, a_{n-2}, \ldots, a_2, a_1, a_n)$. □

Lemma 21 will be very useful in reducing the amount of computation that needs to be done to compute $D(n)$ or $R(n)$ since we can break the whole set of functions down into these classifications.

For instance, when counting $R(7)$, instead of counting all $(0,0,k,0,0,0,0)$ for $k = 1$ to 35, we can just count the profiles up to $k = 17$. Part **(B)** enables us to refer back to $R(6)$ when considering profiles with a nonzero entry in the first position. The most flexible is **(C)**, which effectively cuts all computation time in half.

3.2 Profile Generation

Sequence `A007695` on the OEIS gives the number of profile vectors for any n, and also outlines an algorithm that can be used to compute this number [27]. The references in the OEIS are unpublished and hard to follow; we present the algorithm as Algorithm 5 and prove that it is correct. This algorithm uses a dynamic programming strategy, where the matrix $K(r,x)$ is built up from the previous values $K(r-1,x)$.

Lemma 22. *For $0 < r \leq n$ and $0 < x \leq \binom{n}{\lfloor n/2 \rfloor}$, the (r,x)-th entry of the matrix K output by Algorithm 5 encodes the smallest number of $(r-1)$-sets that can be comparable to any of x number of r-sets.*

Proof. Assume first that $r = 1$, or $r - 1 = 0$. Since $K(0,x) = 0$ for any $x > 0$, Line 16 of the algorithm gives

$$K(r,x) = K(0, x-k-1) + \binom{k-1}{0}$$

$$K(r,x) = 1.$$

Algorithm 5: Generating all profiles of MBFs on n variables.

Input: n
Output: $P(n)$, the list of profiles of MBFs on n variables
initialize $C := \texttt{zeros}\left(n+1, \binom{n}{\lfloor n/2 \rfloor}+1\right)$;
$K := C$, $C(0,0) = 1$, $C(0,1) = 1$, $s = 2$;
initialize $P(n) := \texttt{zeros}\,(n+1, n)$;
set the first column of $P(n)$ to the vector $[0\ 1\ 2\ \cdots\ n]^T$;
$\texttt{total} := n+1$;
for $r = 1$ **to** n **do**
  $d \leftarrow s$, $k \leftarrow r$, $j \leftarrow 0$, $s \leftarrow 0$;
  $x_{\max} = \binom{n}{r}$;
  for $x = 0$ **to** $x_{\max}$ **do**
    if $x \geq \binom{k}{r}$ **then**
      $k \leftarrow k+1$;
    end
    if $x = 0$ **then**
      $K(r,x) = 0$;
    else
      $K(r,x) = K(r-1, x-\binom{k-1}{r}) + \binom{k-1}{r-1}$;
    end
    while $j < K(r,x)$ **do**
      $d \leftarrow d - C(r-1, j)$;
      $j \leftarrow j+1$;
    end
    $C(r,x) = d$;
    $s \leftarrow s+d$;
  end
  if $r \neq 1$ **then**
    `recent` = last $C(r,0) - C(r-1,0)$ rows of $P(n)$;
    for $x = 1$ **to** $x_{\max}$ **do**
      `candidates` = rows of `recent` with $(r-1)$-st entry at least $K(r,x)$. ;
      Subtract $K(r,x)$ from the $(r-1)$-st column of `candidates` ;
      Add x to the r-th column of `candidates` ;
      Append `candidates` to $P(n)$;
      Update `total` $\leftarrow$ `total` +size(`candidates`). ;
    end
  end
end
Output $P(n)$. ;

This is true since the empty set is comparable to any number of singleton sets, and there is only one empty set.

Now assume that $r > 1$. Again, Line 16 of the algorithm gives the recursion step: $K(r,x) = K(r-1, x - \binom{k-1}{r}) + \binom{k-1}{r-1}$. We consider two cases, one where the value of k was updated in Line 11 and one where it was not.

Case 1: If k was updated in Line 10, then x is exactly equal to $\binom{k}{r}$ before k was incremented by 1. This means, at Line 16, x is equal to $\binom{k-1}{r}$, and hence $K(r-1, x - \binom{k-1}{r}) = K(r-1,0) = 0$. Now if we have the exact collection $\mathcal{U} = \binom{\{1,2,\ldots,k-1\}}{r}$, then the number of $(r-1)$-sets contained in our collection of x r-sets must be equal to $\binom{k-1}{r-1}$, counting all $(r-1)$-subsets of $[k-1] = \{1,2,\ldots,k-1\}$.

Since any other collection of x r-sets must contain at least k elements, then the number we obtained when we considered $\mathcal{U} = \binom{\{1,2,\ldots,k-1\}}{r}$ must have been the lower bound for any such collection. Thus, $K(r,x) = K(r-1, x - \binom{k-1}{r}) + \binom{k-1}{r-1}$ must be the lower bound for the number of $(r-1)$-sets contained in x r-sets.

Case 2: If k was not updated in Line 10, then $\binom{k-1}{r} < x < \binom{k}{r}$. Consider the family of sets $\mathcal{A} = \mathcal{U} \cup \mathcal{V}$ where $\mathcal{U}$ is the set of all r-subsets of $[k-1]$, and $\mathcal{V}$ contains $x - \binom{k-1}{r}$ other sets, all of which contain the element k.

The number of $(r-1)$-sets contained in $\mathcal{U}$ is just equal to $\binom{k-1}{r-1}$, that is, all possibilities of taking $r-1$ elements from $[k-1]$.

As for $\mathcal{V}$, since $\mathcal{U}$ already contains all $(r-1)$-subsets of $[k-1]$, we will only count the number of $(r-1)$-sets comparable to $\mathcal{V}$ which contain the element k. By removing k from all of the sets in $\mathcal{V}$, we can see that this number is bounded below by the number of $(r-2)$-sets that must be contained in any collection of $x - \binom{k-1}{r}$ $(r-1)$-sets, or by induction, the value $K(r-1, \binom{k-1}{r})$.

Hence, the number of $(r-1)$-sets that are necessarily contained in any collection of x r-sets must be bounded below by $K(r,x) = K(r-1, \binom{k-1}{r}) + \binom{k-1}{r-1}$, completing the proof. $\square$

Theorem 23. *The list $P(n)$ in Algorithm 5 contains all profiles of monotone Boolean functions on n variables.*

Proof. We perform induction on the rightmost nonzero entry in a profile vector.

When $P(n)$ is initialized, the empty profile and all profiles with a single nonzero entry in the first position are included. This is just the collection of all MBFs with only 1-sets as minimal terms, and there are n such profiles as there are n such sets in $2^{[n]}$.

Now assume that $P(n)$ contains all profiles with the rightmost nonzero entry in the r-th position.

First of all, note that when $x = 0$, the conditional in Line 18 fails, and so $C(r,0) = d$, which was most recently updated to the value of s, the running total of all profiles so far. This means that $C(r,0) - C(r-1,0)$ is the number of new profiles added when iterating in the $(r-1)$-st row. From Line 26, we can conclude that `recent` contains exactly the profiles with rightmost nonzero entry in the $(r-1)$-st position.

The loop starting at Line 28 looks at $K(r,x)$, which by the previous lemma tells us how many $(r-1)$-sets are equivalent to x r-sets. Then all the profiles which have $(r-1)$-st entry *at least* $K(r,x)$ will be taken – this is `candidates`. The next few lines do a substitution, replacing this number of $(r-1)$-sets by x in the r-th entry. This new set of profile vectors is then appended into the existing list, and values are updated.

Note that we still have to prove the fact that the variable s keeps track of how many profiles have been generated already. Since s is at each iteration incremented by d, which is in turn $C(r,x)$, we just have to prove that $C(r,x)$ encodes the number of profiles such that the rightmost nonzero entry is an x in the rth position. But this is apparent from the loop starting at Line 18, since we are forcing j to be larger than $K(r,x)$, so that from the previous row, we are only looking at profiles where the $(r-1)$-st entry is at least $K(r,x)$. This allows us to make the substitution we describe above in the loop starting Line 28. □

This algorithm does not generate the functions themselves, but only lists the possible profile vectors an n-variable MBF may have. This is why we the "equivalences" discussed in the previous proofs are not exact ones, but rather are thresholds above which it is possible

to perform such a comparison to obtain smaller-sized sets.

Example 24. Here is an example to show what the above algorithm does: Suppose we wanted to generate all profiles of 5-variable monotone Boolean functions which have two 3-sets and no 4- or 5-sets. The fact that $K(3,2) = 5$ means that two 3-sets contains at least five 2-sets.

For instance, the collection $\{1,2,3\},\{1,2,4\}$ contains the sets $\{1,2\},\{1,3\},\{2,3\},\{1,4\},\{2,4\}$, a total of five, while the collection $\{1,2,3\},\{1,4,5\}$ contains $\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{4,5\}$, a total of six.

The algorithm would then look at all profiles generated during the previous r-loop, which in this case would be

$(0,1,0,0,0)$	$(1,2,0,0,0)$	$(0,4,0,0,0)$	$(\underline{1,6,0,0,0})$
$(1,1,0,0,0)$	$(2,2,0,0,0)$	$(1,4,0,0,0)$	$(\underline{0,7,0,0,0})$
$(2,1,0,0,0)$	$(0,3,0,0,0)$	$(\underline{0,5,0,0,0})$	$(\underline{0,8,0,0,0})$
$(3,1,0,0,0)$	$(1,3,0,0,0)$	$(\underline{1,5,0,0,0})$	$(\underline{0,9,0,0,0})$
$(0,2,0,0,0)$	$(2,3,0,0,0)$	$(\underline{0,6,0,0,0})$	$(\underline{0,10,0,0,0})$,

of which the underlined ones have the required minimum of five 2-sets. The last step would be to subtract 5 from the second entry of these vectors, and to add 2 to the third entry, resulting in the following profiles with exactly 2 3-sets:

$(0,0,2,0,0)$	$(0,2,2,0,0)$
$(1,0,2,0,0)$	$(0,3,2,0,0)$
$(0,1,2,0,0)$	$(0,4,2,0,0)$
$(1,1,2,0,0)$	$(0,5,2,0,0)$.

Using the algorithm, we can compute the number of profiles of MBFs on n variables, which is Sequence A007695 on the OEIS:

n	Number of profiles	n	Number of profiles
0	2	5	96
1	3	6	553
2	5	7	5461
3	10	8	100709
4	26	9	3718354

Table 3.1: Number of profiles for each n, from $n = 0$ to $n = 9$.

3.3 Using Profiles to Generate Functions

Using the techniques in the previous section, we are able to generate functions in both $D(n)$ and $R(n)$ using a strategy similar to the technique of Shmulevich et al. outlined in Section 2.2.4.

Given a profile $(a_1, a_2, \ldots, a_n)$, one way to compute $(a_1, \ldots, a_n)_D$ is to take the a_i entries which are nonzero, then perform disjunctions one by one, making sure that subsumption does not occur.

The main difference with Shmulevich's technique is that we are breaking down the set of MBFs into smaller classes by considering the 5461 profiles for $n = 7$ instead of just "all MBFs with k minimal terms," where k is at most 35. This further restricts the search space, so we will generate many fewer redundant functions. This immediately presents a challenge – finding the best way to work through the profiles one by one in order to use the available computing time most efficiently.

Moreover, we still have to consider the process of eliminating equivalent functions in order to obtain $R(7)$, or at least the R-value of each profile. At present it appears difficult to extend this technique to computing $R(8)$. This is because it requires generating, rather than merely counting, a nontrivial fraction of the profiles.

To explain the algorithm further, let us take the following sequence of profiles:

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

The algorithm starts at the zero profile, which has only the function $f = 0$, by our convention. (We ignore the all-ones function since nothing can be added to it anymore.) To compute the number of functions with the profile following it, $(0,0,1,0,0,0,0)_D$, we consider all three-sets which are not subsumed by $f = 0$. As this is just the set of all functions with exactly one three-set as its minimal term, we add all $\binom{7}{3} = 35$ of them to our list. Now, as we are computing $(0,0,1,0,0,0,0)_R$, we would have to do the additional computations of finding the least representatives of each of these 35 functions, and eliminating duplicates. By doing this, we are left with only the function with minimal term $\{3,4,5\}$, which has a 32-bit integer representation of $(15,15,15,15)$, and hence we get $(0,0,1,0,0,0,0)_R = 1$.

To proceed to $(0,0,2,0,0,0,0)_R$, we now consider all three-sets which are not subsumed by the set $\{3,4,5\}$ – all the remaining 34 three-sets. Taking disjunctions, we get the 34 functions with minimal terms $\{3,4,5\}$ plus another 3-set, and after getting the least representatives of each one and eliminating duplicates, we are left with three nonequivalent MBFs with the profile $(0,0,2,0,0,0,0)$:

$$\begin{aligned}(0,0,2,0,0,0,0)_R \text{ contains } &(63,63,63,63)\\ &(855,855,855,855)\\ &(218455,196611,218455,196611)\end{aligned}$$

They are the functions with minimal terms $\{\{2,4,5\},\{3,4,5\}\}$, $\{\{2,3,5\},\{1,4,5\}\}$, and $\{\{2,3,4\},\{1,5,6\}\}$. Note that every other MBF which has two 3-sets as minimal terms is

equivalent to one of these three, which can be determined by the number of elements in the intersection of the two sets (either two, one, or none).

Continuing the computations, we obtain:

$$(0,0,1,0,0,0,0)_R = 1 \qquad (0,0,2,2,0,0,0)_R = 94$$
$$(0,0,2,0,0,0,0)_R = 3 \qquad (0,1,2,2,0,0,0)_R = 437$$
$$(0,0,2,1,0,0,0)_R = 14$$

With the goal of enumerating $R(7)$, we use Lemma 21 to eliminate profiles we do not have to compute, leaving us with a significantly smaller number of profiles which are "essential." However, we will have to compute some profiles more than once, as some profiles whose R-value can be obtained by symmetry are essential as well.

Moreover, we also generate all MBFs which reside in exactly one level, i.e. uniform-rank MBFs. This way, we do not have to start all over again from $(0,0,0,0,0,0,0)$ whenever we begin computation on a new branch. In fact, we try to generate as many lists as we can in order to save computing time when going through profiles with a large number of terms. We generate two-level profiles after we generate uniform-level profiles, and then branch off to three-level profiles.

For instance, we first generated the list of NMBFs with profiles $(0,0,a_3,0,0,0,0)$, then from each of these lists we are building up to all functions with profiles of the form $(0,0,a_3,a_4,0,0,0)$, then those with profiles $(0,a_2,a_3,a_4,0,0,0)$. So a typical computation would be, once we have generated $(0,0,5,0,0,0,0)$, is to generate all $(0,0,5,a_4,0,0,0)$ for $a_4 = 1$ to $a_4 = 25$. Then, we pick the starting point $(0,0,5,3,0,0,0)$ to count all functions in $(0,a_2,5,3,0,0,0)$ for $a_2 = 1$ to $a_2 = 7$.

It should be noted that the profiles which are solely contained in the third and fourth levels are the most difficult to compute, as the presence of any 2-set or 5-set in the MBF drastically reduces the number of 3-sets or 4-sets that can be added on as a minimal term without being subsumed. Specifically, a 2-set is contained in five different 3-sets, and $\binom{5}{2} = 10$ different 4-sets. On the other hand, the presence of a 3-set does not reduce the number of 4-sets we have to check for incomparability, as out of 35 4-sets, only four contain any given

3-set.

The largest profile for $R(7)$ computed so far is that of $(0,0,13,2,0,0,0)_R = 3136686$. As an illustration, we give in Table 3.3 the list of 95 profiles in $R(5)$ and the number of functions in each one. Note the list does not include the all-ones function. We also give corresponding values for $D(6)$ and $R(6)$ in Appendix B.

As a byproduct of the calculations done for $R(7)$, we also extended the known values for the sequences $R_k(n)$, which we are submitting into the OEIS. The corresponding sequences for $D_k(n)$ are `A051112` to `A051118` [27]. We have the new values in Table 3.2 which we have computed so far.

Out of the estimated $D(7)/7! \sim 479$ million equivalence classes in $R(7)$, we have counted over 200 million so far.

k	$R_k(5)$	$R_k(6)$	$R_k(7)$
2	13	22	34
3	30	84	202
4	49	287	**1321**
5	48	787	**8626**
6	34	1661	**50961**
7	18	2630	**253104**
8	7	3164	**1025322**
9	2	2890	**3365328**
10	2	2159	**9005678**
11	X	1327	**19850932**

Table 3.2: Partial list of values $R_k(n)$, where values in boldface are new results.

Profile	#	Profile	#	Profile	#	Profile	#
(0,0,0,0,0)	1	(0,9,0,0,0)	1	(1,0,3,0,0)	1	(0,2,0,1,0)	1
(1,0,0,0,0)	1	(0,10,0,0,0)	1	(0,1,3,0,0)	6	(0,3,0,1,0)	1
(2,0,0,0,0)	1	(0,0,1,0,0)	1	(0,2,3,0,0)	6	(0,4,0,1,0)	1
(3,0,0,0,0)	1	(1,0,1,0,0)	1	(0,3,3,0,0)	4	(0,0,1,1,0)	1
(4,0,0,0,0)	1	(2,0,1,0,0)	1	(0,4,3,0,0)	1	(0,1,1,1,0)	1
(5,0,0,0,0)	1	(0,1,1,0,0)	2	(0,0,4,0,0)	6	(0,2,1,1,0)	1
(0,1,0,0,0)	1	(1,1,1,0,0)	1	(1,0,4,0,0)	1	(0,0,2,1,0)	2
(1,1,0,0,0)	1	(0,2,1,0,0)	4	(0,1,4,0,0)	6	(0,1,2,1,0)	1
(2,1,0,0,0)	1	(1,2,1,0,0)	1	(0,2,4,0,0)	4	(0,0,3,1,0)	3
(3,1,0,0,0)	1	(0,3,1,0,0)	6	(0,3,4,0,0)	1	(0,1,3,1,0)	1
(0,2,0,0,0)	2	(1,3,1,0,0)	1	(0,4,4,0,0)	1	(0,0,4,1,0)	2
(1,2,0,0,0)	2	(0,4,1,0,0)	6	(0,0,5,0,0)	6	(0,0,5,1,0)	1
(2,2,0,0,0)	1	(0,5,1,0,0)	4	(0,1,5,0,0)	4	(0,0,6,1,0)	1
(0,3,0,0,0)	4	(0,6,1,0,0)	2	(0,2,5,0,0)	1	(0,0,0,2,0)	1
(1,3,0,0,0)	3	(0,7,1,0,0)	1	(0,0,6,0,0)	6	(0,1,0,2,0)	1
(2,3,0,0,0)	1	(0,0,2,0,0)	2	(0,1,6,0,0)	2	(0,0,1,2,0)	1
(0,4,0,0,0)	6	(1,0,2,0,0)	1	(0,0,7,0,0)	4	(0,0,2,2,0)	1
(1,4,0,0,0)	2	(0,1,2,0,0)	4	(0,1,7,0,0)	1	(0,0,3,2,0)	1
(0,5,0,0,0)	6	(1,1,2,0,0)	1	(0,0,8,0,0)	2	(0,0,0,3,0)	1
(1,5,0,0,0)	1	(0,2,2,0,0)	7	(0,0,9,0,0)	1	(0,0,1,3,0)	1
(0,6,0,0,0)	6	(0,3,2,0,0)	6	(0,0,10,0,0)	1	(0,0,0,4,0)	1
(1,6,0,0,0)	1	(0,4,2,0,0)	4	(0,0,0,1,0)	1	(0,0,0,5,0)	1
(0,7,0,0,0)	4	(0,5,2,0,0)	1	(1,0,0,1,0)	1	(0,0,0,0,1)	1
(0,8,0,0,0)	2	(0,0,3,0,0)	4	(0,1,0,1,0)	1	**TOTAL**	**209**

Table 3.3: Number of nonequivalent five-variable MBFs by profile.

Chapter 4

Conclusion and Future Work

This thesis looked at the Dedekind numbers and the various ways that they can be counted. We mentioned that monotone Boolean functions play a significant role in many fields in mathematics, especially in computational biology, image processing, and learning theory.

We propose a strategy for counting nonequivalent monotone Boolean functions by breaking the problem into parts based on the profiles of the functions. We described how to generate all profiles of n-variable MBFs.

This computation of $R(7)$ is still ongoing, however from partial results we have already extended a series of sequences on *The Online Encyclopedia of Integer Sequences* [27]. Specifically, we have computed the terms for $n = 7$ for the sequences of NMBFs which count $R_k(n)$ for $k = 4, 5, \ldots, 11$, and look to finish this for up to $k = 35$.

It is not clear if this calculation can be leveraged to help compute $D(9)$. It is tempting to try to combine a complete generation of $R(7)$ with the enumeration strategy of Section 2.2.3, but it is not clear how this would work.

Bibliography

[1] C. Berge. *Hypergraphs - Combinatorics of Finite Sets*, volume 45 of *North Holland Mathematical Library*. Elsevier, 1989.

[2] J. Berman and P. Köhler. Cardinalities of finite distributive lattices. *Mitt. Math. Sem. Giessen*, (Heft 121):103–124, 1976.

[3] S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag, 1981.

[4] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Chapman and Hall, fourth edition, 2004.

[5] R. Church. Numerical analysis of certain free distributive structures. *Duke Mathematical Journal*, 6(3):732–734, 1940.

[6] R. Church. Enumeration by rank of the free distributive lattice with 7 generators. *Notices of the American Mathematical Society*, (11):724, 1965.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[8] R. Dedekind. Über Zerlegungen von Zahlen durch ihre grossten gemeinsamen Teiler. In *Ges. Werke, Vol. 2*, pages 103–148, 1897.

[9] T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156:2035–2049, 2008.

[10] K. Engel. *Sperner Theory*. Cambridge University Press, 1997.

[11] J.D. Farley. Breaking Al Qaeda cells: A mathematical analysis of counterterrorism operations (A guide for risk assessment and decision making). *Studies in Conflict and Terrorism*, 26:399–411, 2003.

[12] R. Fidytek, A. Mostowski, R. Somla, and A. Szepietowski. Algorithms counting monotone boolean functions. *Information Processing Letters*, 79:203–209, 2001.

[13] M. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21:618–628, 1996.

[14] B. Ganter and R. Wille. Applied lattice theory: Formal concept analysis. `http://wwwbib.mathematik.tu-darmstadt.de/Math-Net/Preprints/Listen/pp97.html`, 1997.

[15] U. Haus, S. Klamt, and T. Stephen. Computing knock-out strategies in metabolic networks. *J. Comput. Biol.*, 15(3):259–268, 2008.

[16] H. Ito, M. Kobayashi, and G. Nakamura. Semi-distance codes and steiner systems. *Graph. Comb.*, 23:283–290, February 2007.

[17] G. Kilibarda and V. Jovovic. On the number of monotone boolean functions with fixed number of lower units (in Russian). *Intellektualnye sistemy*, 7(1-4):193–217, 2003.

[18] A. Kisielewicz. A solution of Dedekind's problem on the number of isotone boolean functions. *J. Reine Angew. Math.*, 386:139–144, 1988.

[19] S. Klamt and E.D. Gilles. Minimal cut sets in biochemical reaction networks. *Bioinformatics*, 20(2):226–234, 2004.

[20] D. Kleitman. On Dedekind's problem: The number of monotone boolean functions. *Proc. Amer. Math. Soc.*, 21:677–682, 1969.

[21] A. Korshunov. Monotone boolean functions. *Russian Mathematical Surveys*, 58(5):929–1001, 2003.

[22] E. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 652–658, New York, NY, USA, 1999. ACM.

[23] E. Reck. Dedekindś contributions to the foundations of mathematics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2011 edition, 2011.

[24] F. Riquelme and A. Polyméris. On the complexity of the decisive problem in simple and weighted games. *Electronic Notes in Discrete Mathematics*, 37:21–26, 2011.

[25] I. Shmulevich. Computational learning theory. `http://personal.systemsbiology.net/ilya/LEARN.htm`.

[26] I. Shmulevich, T. M. Sellke, M. Gabbouj, and E. J. Coyle. Stack filters and free distributive lattices. In *Proceedings of the 1995 IEEE Workshop on Nonlinear Signal and Image Processing*, pages 927–930, Halkidiki, Greece, 1995.

[27] N. J. A. Sloane. The online encyclopedia of integer sequences. `http://oeis.org`, 2011.

[28] M. Ward. Note on the order of free distributive lattices. *Bulletin of the American Mathematical Society*, (52):423, 1946.

[29] E. Weisstein. Mathworld - A Wolfram Web Resource. `http://mathworld.wolfram.com/Antichain`. Antichain.

[30] D. Wiedemann. A computation of the eight Dedekind number. *Order*, 8(1):5–6, 1991.

Appendix A

$R(7)$ Profile Values

In this appendix we list some of the profiles that have been computed, their respective values, and the time it took to compute them. Specifically, we give the totals for each one-level and two-level profile we have so far; we omit those profiles which have a nonzero entry in the first or sixth position in the profile vector as these ones can be derived from the list of profiles of $R(6)$, and profile $(0,0,0,0,0,0,1)$ which is trivial. Currently we have enumerated over 90% of all 7-variable profiles.

Calculations were done primarily on the SFU-Surrey Optima Cluster, with a few others on the IRMACS Computational Cluster, hence the CPU times specified here do not come from a consistent source. Moreover, these calculations were also done using a version of the code which used binary search to do lookups and insertions. We have since transitioned to a new version of the code that implements the hash tables described in Section 2.1.2.

All times given are in CPU seconds.

A.1 Profiles $(0,a_2,0,0,0,0,0)$ and $(0,0,a_3,0,0,0,0)$

Note that these are sequences `A008406` and `A092337` from [27]. The times given measure how long it took to generate that particular $(0,a_2,0,0,0,0,0)_R$ from the previous $(0,a_2-1,0,0,0,0,0)_R$ (as for $(0,0,a_3,0,0,0,0,0)$). We can get the values for the "complementary

profiles" via Lemma 21, as well as the values for all the fourth-level and fifth-level profiles.

a_2, $21-a_2$	# NMBFs	Time
1, 20	1	0.9
2, 19	2	0.87
3, 18	5	1.61
4, 17	10	3.83
5, 16	21	7.21
6, 15	41	14.28
7, 14	65	26.11
8, 13	97	38.69
9, 12	131	53.44
10, 11	148	66.74

a_3, $35-a_3$	# NMBFs	Time
1, 34	1	1.47
2, 33	3	1.27
3, 32	10	3.68
4, 31	38	11.91
5, 30	137	44.26
6, 29	509	154.32
7, 28	1760	555.93
8, 27	5557	1856.04
9, 26	15709	5668.99
10, 25	39433	15584.74
11, 24	87659	38884.67
12, 23	172933	86818.52
13, 22	303277	175482.74
14, 21	473827	305039.96
15, 20	660950	461243.95
16, 19	824410	670002.13
17, 18	920446	972417.16

Table A.1: Values and times for second-level and third-level profile generation.

A.2 Two-level profiles

The hardest to compute are the profiles of the form $(0,0,a_3,a_4,0,0,0)$. We have not yet finished computing these profiles; here we list the ones that are done. Note that by symmetry, we do not list those with $a_3 < a_4$.

We also list all two-level profiles of the form $(0,a_2,a_3,0,0,0,0)$ and $(0,a_2,0,a_4,0,0,0)$.

a_3	a_4	# NMBFs	Time
1	1	3	1.22
2	1	14	3.01
3	1	72	8.45
4	1	347	29.15
5	1	1522	92.91
6	1	5945	347.7
7	1	20033	1049.32
8	1	58184	2386.9
9	1	145962	6003.25
10	1	317525	23743.61
11	1	601963	80674.8
12	1	998926	242786.13
13	1	1456007	94280.66
14	1	1869011	501544.04
15	1	2116691	504051.81
16	1	2116691	520705.1
17	1	1869011	501355
18	1	1456007	376161.82
19	1	998926	188481.83
20	1	601963	80644.97
21	1	317525	19672.82
22	1	145962	9208.94
23	1	58184	142734.56
24	1	20033	28982.88
25	1	5945	6278.33
26	1	1522	1366.88
27	1	347	283.12
28	1	72	51.05
29	1	14	7.96
30	1	3	1.06
31	1	1	0.12

a_3	a_4	# NMBFs	Time
2	2	94	15.18
3	2	541	102.17
4	2	2741	250.61
5	2	11637	966.04
6	2	41377	4019.39
7	2	123184	13556.11
8	2	309711	32587.91
9	2	662750	95635.1
10	2	1215801	483961.3
11	2	1923318	1195477.02
12	2	2635180	2323954.54
13	2	3136686	281464.53
14	2		
15	2	2931257	1261444.58
16	2		
17	2		
18	2	926868	208553.34
19	2	472090	98880.63
20	2	206263	70011.67
21	2	76822	16592.81
22	2	24238	3585.54
23	2	6454	751.72
24	2	1456	1559.62
25	2	281	276.72
26	2	48	39.4
27	2	8	4.48
28	2	1	0.34

a_3	a_4	# NMBFs	Time
3	3	3250	509.39
4	3	15659	2951.82
5	3	60223	8048.28
6	3	187967	26864.23
7	3	482449	141991.39
8	3	1031741	554402.88
9	3	1857424	1706461.3
10	3		
11	3		
12	3		
13	3		
14	3		
15	3		
16	3		
17	3		
18	3	310937	69428.42
19	3	114241	28357.83
20	3	35126	7858.44
21	3	8992	1648.56
22	3	1925	314.27
23	3	347	53.75
24	3	56	42.28
25	3	8	4.53
26	3	1	0.34

Table A.2: Middle-level profiles with $a_3 > 0, a_4 = 1, 2, 3$.

a_3	a_4	# NMBFs	Time
4	4	67871	12261.94
5	4	228309	42298.72
6	4	612342	438418.61
7	4	1334055	2318724.13
18	4	64460	16615.71
19	4	17329	4798.12
20	4	3934	810.92
21	4	776	152.21
22	4	136	24.73
23	4	23	3.8
24	4	4	2.05
25	4	1	0.22

a_3	a_4	# NMBFs	Time
5	5	659997	375920.9
6	5	1501560	1435924.08
18	5	10513	3470.75
19	5	2474	814.08
20	5	561	122.6
21	5	133	26.06
22	5	30	5.19
23	5	8	4.03
24	5	2	0.78
25	5	1	0.1

a_3	a_4	# NMBFs	Time
18	6	1445	566.55
19	6	281	104.24
20	6	47	11.8
21	6	8	1.78
22	6	1	0.37
18	7	143	51.22
19	7	19	6.73
20	7	1	0.28
18	8	15	5.6
19	8	2	0.58
18	9	4	0.9
19	9	1	0.15

Table A.3: Partial results for middle-level profiles with $a_3 > 0, a_4 = 4, 5, \ldots, 9$.

a_2	a_3	#	Time
1	1	2	2.18
2	1	7	3.93
3	1	21	8.16
4	1	53	15.29
5	1	115	28.6
6	1	219	51.53
7	1	341	84.06
8	1	449	128.86
9	1	494	185.85
10	1	449	249.96
11	1	341	312.82
12	1	219	369.18
13	1	115	413.23
14	1	53	443.48
15	1	21	461.08
16	1	7	470.82
17	1	2	475.15
18	1	1	477.03
1	2	10	4.71
2	2	38	11.53
3	2	118	27.11
4	2	299	55.21
5	2	604	99.68
6	2	989	159.96
7	2	1295	218.14
8	2	1367	266.87
9	2	1170	302.05
10	2	805	328.32
11	2	448	355.76
12	2	202	388.32
13	2	75	419.92
14	2	22	445.31
15	2	6	461.36
16	2	1	471.05
1	3	43	16.91
2	3	183	50.59
3	3	555	128.35
4	3	1272	266.35
5	3	2219	444.87
6	3	3004	609.19
7	3	3139	678.28
8	3	2572	638.93
9	3	1650	538.84
10	3	824	445.74
11	3	323	401.04
12	3	103	401.68
13	3	25	423.2
14	3	5	445.62
15	3	1	461.62

a_2	a_3	#	Time
1	4	199	67.57
2	4	811	231.24
3	4	2210	580.83
4	4	4339	1111.24
5	4	6245	1624.58
6	4	6744	1864.5
7	4	5492	1689.72
8	4	3405	1261.46
9	4	1618	830.05
10	4	586	547.94
11	4	170	428.28
12	4	41	407.19
13	4	9	424.16
14	4	2	445.6
15	4	1	461.78
1	5	806	292.37
2	5	3020	993.44
3	5	7109	2287.24
4	5	11563	3843.98
5	5	13346	4707.73
6	5	11254	4449.96
7	5	6969	3287.9
8	5	3203	1990.27
9	5	1097	1076.36
10	5	287	609.68
11	5	57	440.18
12	5	10	408.94
13	5	1	424.3
1	6	2977	1166.13
2	6	9643	3706.46
3	6	18883	7503.01
4	6	24744	10631.99
5	6	22473	10844.22
6	6	14536	8410.99
7	6	6748	5080.26
8	6	2279	2591.7
9	6	581	1222.06
10	6	110	634.71
11	6	18	443.91
12	6	2	409.35
1	7	9510	4271.41
2	7	25843	12122.74
3	7	41176	20689.65
4	7	42740	24297.08
5	7	29977	20417.63
6	7	14640	13033.91
7	7	5028	6622.66
8	7	1259	2969.9
9	7	239	1289.12
10	7	36	643.87
11	7	7	445.37
12	7	1	409.57

a_2	a_3	#	Time
1	8	26314	14044.81
2	8	58617	33554.36
3	8	74633	48217.86
4	8	60345	46500.62
5	8	32192	32148.14
6	8	11720	17211.42
7	8	2987	7670.79
8	8	558	3169.05
9	8	83	1314.82
10	8	10	646.38
11	8	2	445.99
1	9	62852	39626.24
2	9	112940	81732.29
3	9	113123	97845.45
4	9	70122	76078
5	9	28008	43956
6	9	7578	20694.93
7	9	1448	8341.99
8	9	212	3238.14
9	9	25	1347.24
10	9	3	669.1
11	9	1	460.15
1	10	130228	102687.9
2	10	185970	19046.2
3	10	144281	22167.04
4	10	67448	12736.89
5	10	19949	4619.75
6	10	4057	1075.31
7	10	615	183.2
8	10	74	23.01
9	10	9	2.26
10	10	2	0.4
11	10	1	0.1
1	11	234746	43095.31
2	11	262724	152259.26
3	11	155305	127955.06
4	11	53849	46662.01
5	11	11673	6491.5
6	11	1846	957.14
7	11	227	129.86
8	11	20	11.13
9	11	1	0.48
1	12	369736	22017.68
2	12	319639	39525.49
3	12	141496	23294.27
4	12	35745	7743.52
5	12	5719	1477.21
6	12	776	217.3
7	12	84	26.11
8	12	5	1.64

Table A.4: Two-level profiles where $a_2 > 0$ and $a_3 = 1, 2, \ldots, 12$.

a_2	a_3	#	Time
1	13	510312	76763.32
2	13	335487	127572.36
3	13	109069	60686.65
4	13	19763	7423.41
5	13	2386	999.93
6	13	314	147.17
7	13	26	11.42
8	13	1	0.54
1	14	618609	157278.07
2	14	303972	111095.75
3	14	71043	26390.56
4	14	9099	1983.98
5	14	876	193.27
6	14	129	28.24
7	14	8	2.05
1	15	659478	52662.91
2	15	237654	27990.91
3	15	39009	6828.45
4	15	3515	822.52
5	15	290	78.82
6	15	50	11.75
7	15	2	0.75
1	16	618609	45764.57
2	16	160040	18548.9
3	16	18007	3320.16
4	16	1146	296.06
5	16	93	20.87
6	16	23	3.86
7	16	1	0.1
1	17	510312	587856.85
2	17	92532	10499.29
3	17	6948	1156.75
4	17	326	71.57
5	17	26	5.58
6	17	7	3.62
1	18	369736	40675.73
2	18	45759	8793.74
3	18	2253	713.13
4	18	80	30.7
5	18	7	3.18
6	18	3	0.91

a_2	a_3	#	Time
1	19	234746	25688.81
2	19	19281	4836.84
3	19	618	361.61
4	19	19	14.22
5	19	2	0.64
6	19	1	0.26
1	20	130228	16284.07
2	20	6891	2328.13
3	20	146	153.22
4	20	4	2.31
5	20	1	0.1
6	20	1	0.06
1	21	62852	4170.92
2	21	2090	418.37
3	21	30	11.77
4	21	1	0.18
1	22	26314	2703.68
2	22	545	156.35
3	22	7	2.96
1	23	9510	1103.03
2	23	123	51.92
3	23	1	0.62
1	24	2977	414.28
2	24	25	11.9
1	25	806	213.89
2	25	5	4.13
1	26	199	79.27
2	26	1	0.96
1	27	43	21.12
1	28	10	4.64
1	29	2	0.73
1	30	1	0.08

Table A.5: Two-level profiles where $a_2 > 0$ and $a_3 = 13, 14, \ldots, 30$.

a_2	a_4	#	Time
1	1	2	0.6
2	1	6	1.01
3	1	15	2.78
4	1	30	6.42
5	1	53	11.74
6	1	80	18.79
7	1	94	25.45
8	1	94	26.54
9	1	80	23.26
10	1	53	16.91
11	1	30	9.37
12	1	15	4.26
13	1	6	1.61
14	1	2	0.44
15	1	1	0.08
1	2	8	1.54
2	2	24	2.94
3	2	52	7.91
4	2	88	15.38
5	2	116	23.19
6	2	121	26.88
7	2	98	24.15
8	2	64	16.49
9	2	34	8.74
10	2	14	3.5
11	2	4	1.01
12	2	1	0.16
1	3	28	2.67
2	3	76	6.88
3	3	131	16.79
4	3	165	25.48
5	3	156	28.06
6	3	115	23.11
7	3	65	14.43
8	3	29	6.95
9	3	12	2.5
10	3	3	0.75
11	3	1	0.12
1	4	106	9.05
2	4	222	22.53
3	4	283	42.3
4	4	259	47.66
5	4	179	39.57
6	4	102	24.1
7	4	44	12.07
8	4	18	5.15
9	4	8	1.93
10	4	3	0.75
11	4	1	0.15

a_2	a_4	#	Time
1	5	338	21.64
2	5	519	49.49
3	5	472	69.21
4	5	314	57.97
5	5	169	35.95
6	5	75	17.81
7	5	25	6.97
8	5	8	2.27
9	5	4	0.86
10	5	2	0.39
11	5	1	0.11
1	6	980	51.58
2	6	1034	308.35
3	6	658	187.83
4	6	328	88.55
5	6	142	40.03
6	6	51	13.98
7	6	9	2.21
8	6	1	0.24
1	7	2431	121.78
2	7	1710	930.84
3	7	756	346.89
4	7	299	132.06
5	7	125	49.64
6	7	38	15.65
7	7	4	1.11
1	8	5209	268.97
2	8	2374	1445.67
3	8	724	377.75
4	8	247	120.92
5	8	104	47.78
6	8	30	12.99
7	8	2	0.49
1	9	9520	386.91
2	9	2739	2147.48
3	9	585	337.88
4	9	177	96.09
5	9	79	38.98
6	9	23	10.43
7	9	1	0.13
1	10	14964	636.63
2	10	2658	2175.6
3	10	392	246.45
4	10	111	63.42
5	10	52	26.79
6	10	15	7.44

a_2	a_4	#	Time
1	11	20179	923.6
2	11	2155	1947.76
3	11	222	146.35
4	11	58	34.37
5	11	29	15.28
6	11	9	4.43
1	12	23428	13683.82
2	12	1466	1366.36
3	12	105	69.91
4	12	25	14.79
5	12	13	6.87
6	12	5	2.16
1	13	23428	14124.69
2	13	829	848.36
3	13	42	27.08
4	13	9	4.8
5	13	5	2.33
6	13	2	0.89
1	14	20179	12751.55
2	14	396	405.34
3	14	13	8.06
4	14	3	1.17
5	14	2	0.62
6	14	1	0.25
1	15	14964	9764.05
2	15	158	165.28
3	15	4	1.73
4	15	1	0.2
5	15	1	0.13
6	15	1	0.07
1	16	9520	6306.94
2	16	53	53.57
3	16	1	0.3
1	17	5209	3469.36
2	17	15	13.3
1	18	2431	1603.3
2	18	4	2.59
1	19	980	637.02
2	19	1	0.38
1	20	338	216.36
1	21	106	62.11
1	22	28	15.64
1	23	8	3.11
1	24	2	0.6
1	25	1	0.08

Table A.6: Two-level profiles where $a_2, a_4 > 0$.

Appendix B

Values of $D(6)$ and $R(6)$ for each Profile

We give here the table of values for $D(6)$ and $R(6)$ corresponding to each profile. Note that we omit the a_6 term since it is equal to 0 for all but one profile, which is $(0,0,0,0,0,1)$. This profile has 1 function for both $D(6)$ and $R(6)$. Note that we do get the correct totals of $D(6) = 7828354$ and $R(6) = 16353$.

$a_1a_2a_3a_4a_5$	D	R	$a_1a_2a_3a_4a_5$	D	R	$a_1a_2a_3a_4a_5$	D	R	$a_1a_2a_3a_4a_5$	D	R
0 0 0 0 0	1	1	0 3 1 0 0	4400	15	1 0 4 0 0	1260	6	1 0 9 0 0	60	1
1 0 0 0 0	6	1	1 3 1 0 0	2100	6	2 0 4 0 0	15	1	0 1 9 0 0	171600	297
2 0 0 0 0	15	1	2 3 1 0 0	60	1	0 1 4 0 0	27300	62	0 2 9 0 0	52800	108
3 0 0 0 0	20	1	0 4 1 0 0	9900	26	1 1 4 0 0	2100	6	0 3 9 0 0	5480	18
4 0 0 0 0	15	1	1 4 1 0 0	2100	6	0 2 4 0 0	65175	133	0 4 9 0 0	300	2
5 0 0 0 0	6	1	0 5 1 0 0	15840	38	1 2 4 0 0	990	4	0 5 9 0 0	60	1
6 0 0 0 0	1	1	1 5 1 0 0	1260	4	0 3 4 0 0	85530	169	0 0 10 0 0	184760	352
0 1 0 0 0	15	1	0 6 1 0 0	18480	44	1 3 4 0 0	120	1	1 0 10 0 0	6	1
1 1 0 0 0	60	1	1 6 1 0 0	420	2	0 4 4 0 0	66900	137	0 1 10 0 0	120120	220
2 1 0 0 0	90	1	0 7 1 0 0	15840	38	1 4 4 0 0	30	1	0 2 10 0 0	20130	50
3 1 0 0 0	60	1	1 7 1 0 0	60	1	0 5 4 0 0	31605	74	0 3 10 0 0	860	6
4 1 0 0 0	15	1	0 8 1 0 0	9900	26	0 6 4 0 0	8745	28	0 4 10 0 0	30	1
0 2 0 0 0	105	2	0 9 1 0 0	4400	15	0 7 4 0 0	1350	7	0 5 10 0 0	6	1
1 2 0 0 0	270	2	0 10 1 0 0	1320	6	0 8 4 0 0	135	2	0 0 11 0 0	167960	312
2 2 0 0 0	225	2	0 11 1 0 0	240	2	0 9 4 0 0	15	1	0 1 11 0 0	65520	124
3 2 0 0 0	60	1	0 12 1 0 0	20	1	0 0 5 0 0	15504	43	0 2 11 0 0	5220	16
0 3 0 0 0	455	5	0 0 2 0 0	190	3	1 0 5 0 0	1512	6	0 3 11 0 0	60	1
1 3 0 0 0	720	4	1 0 2 0 0	270	2	0 1 5 0 0	65520	124	0 0 12 0 0	125970	249
2 3 0 0 0	300	3	2 0 2 0 0	90	1	1 1 5 0 0	1260	4	0 1 12 0 0	27300	62
3 3 0 0 0	20	1	0 1 2 0 0	1800	8	0 2 5 0 0	112860	208	0 2 12 0 0	825	5
0 4 0 0 0	1365	9	1 1 2 0 0	1260	4	1 2 5 0 0	180	1	0 0 13 0 0	77520	161
1 4 0 0 0	1260	6	2 1 2 0 0	90	1	0 3 5 0 0	101640	190	0 1 13 0 0	8400	23
2 4 0 0 0	225	2	0 2 2 0 0	7650	23	0 4 5 0 0	51120	103	0 2 13 0 0	60	1
0 5 0 0 0	3003	15	1 2 2 0 0	2340	7	0 5 5 0 0	14364	38	0 0 14 0 0	38760	94
1 5 0 0 0	1512	6	0 3 2 0 0	19200	48	0 6 5 0 0	2220	8	0 1 14 0 0	1800	8
2 5 0 0 0	90	1	1 3 2 0 0	2160	6	0 7 5 0 0	180	1	0 0 15 0 0	15504	43
0 6 0 0 0	5005	21	0 4 2 0 0	31500	72	0 0 6 0 0	38760	94	0 1 15 0 0	240	2
1 6 0 0 0	1260	6	1 4 2 0 0	990	4	1 0 6 0 0	1260	6	0 0 16 0 0	4845	21
2 6 0 0 0	15	1	0 5 2 0 0	35280	80	0 1 6 0 0	120120	220	0 1 16 0 0	15	1
0 7 0 0 0	6435	24	1 5 2 0 0	180	1	1 1 6 0 0	420	2	0 0 17 0 0	1140	7
1 7 0 0 0	720	4	0 6 2 0 0	27300	64	0 2 6 0 0	144540	266	0 0 18 0 0	190	3
0 8 0 0 0	6435	24	0 7 2 0 0	14400	38	0 3 6 0 0	85260	167	0 0 19 0 0	20	1
1 8 0 0 0	270	2	0 8 2 0 0	4950	17	0 4 6 0 0	25950	63	0 0 20 0 0	1	1
0 9 0 0 0	5005	21	0 9 2 0 0	1000	6	0 5 6 0 0	4260	16	0 0 0 1 0	15	1
1 9 0 0 0	60	1	0 10 2 0 0	90	1	0 6 6 0 0	420	2	1 0 0 1 0	30	1
0 10 0 0 0	3003	15	0 0 3 0 0	1140	7	0 0 7 0 0	77520	161	2 0 0 1 0	15	1
1 10 0 0 0	6	1	1 0 3 0 0	720	4	1 0 7 0 0	720	4	0 1 0 1 0	135	2
0 11 0 0 0	1365	9	2 0 3 0 0	60	1	0 1 7 0 0	171600	297	1 1 0 1 0	120	1
0 12 0 0 0	455	5	0 1 3 0 0	8400	23	1 1 7 0 0	60	1	0 2 0 1 0	540	4
0 13 0 0 0	105	2	1 1 3 0 0	2100	6	0 2 7 0 0	138600	252	1 2 0 1 0	180	1
0 14 0 0 0	15	1	0 2 3 0 0	27060	61	0 3 7 0 0	50400	106	0 3 0 1 0	1260	6
0 15 0 0 0	1	1	1 2 3 0 0	2160	6	0 4 7 0 0	8700	25	1 3 0 1 0	120	1
0 0 1 0 0	20	1	0 3 3 0 0	49860	102	0 5 7 0 0	1020	6	0 4 0 1 0	1890	9
1 0 1 0 0	60	1	1 3 3 0 0	900	4	0 6 7 0 0	60	1	1 4 0 1 0	30	1
2 0 1 0 0	60	1	0 4 3 0 0	57660	115	0 0 8 0 0	125970	249	0 5 0 1 0	1890	9
3 0 1 0 0	20	1	1 4 3 0 0	120	1	1 0 8 0 0	270	2	0 6 0 1 0	1260	6
0 1 1 0 0	240	2	0 5 3 0 0	43140	91	0 1 8 0 0	193050	334	0 7 0 1 0	540	4
1 1 1 0 0	420	2	0 6 3 0 0	20700	48	0 2 8 0 0	99495	191	0 8 0 1 0	135	2
2 1 1 0 0	180	1	0 7 3 0 0	6060	19	0 3 8 0 0	20535	51	0 9 0 1 0	15	1
0 2 1 0 0	1320	6	0 8 3 0 0	960	5	0 4 8 0 0	1935	8	0 0 1 1 0	240	2
1 2 1 0 0	1260	4	0 9 3 0 0	60	1	0 5 8 0 0	270	2	1 0 1 1 0	180	1
2 2 1 0 0	180	1	0 0 4 0 0	4845	21	0 0 9 0 0	167960	312			

Table B.1: $D(6)$ and $R(6)$ by profile.

$a_1a_2a_3a_4a_5$	D	R
0 1 1 1 0	1620	5
1 1 1 1 0	360	1
0 2 1 1 0	4680	13
1 2 1 1 0	180	1
0 3 1 1 0	7500	18
0 4 1 1 0	7200	18
0 5 1 1 0	4140	11
0 6 1 1 0	1320	5
0 7 1 1 0	180	1
0 0 2 1 0	1800	8
1 0 2 1 0	450	2
0 1 2 1 0	8910	21
1 1 2 1 0	360	1
0 2 2 1 0	18000	40
0 3 2 1 0	18900	40
0 4 2 1 0	10800	25
0 5 2 1 0	3150	10
0 6 2 1 0	360	1
0 0 3 1 0	8400	23
1 0 3 1 0	600	3
0 1 3 1 0	29700	57
1 1 3 1 0	120	1
0 2 3 1 0	40320	80
0 3 3 1 0	26100	52
0 4 3 1 0	8160	22
0 5 3 1 0	1200	6
0 6 3 1 0	120	1
0 0 4 1 0	27300	62
1 0 4 1 0	450	2
0 1 4 1 0	66825	118
0 2 4 1 0	57960	109
0 3 4 1 0	21300	46
0 4 4 1 0	3420	10
0 5 4 1 0	450	2
0 0 5 1 0	65520	124
1 0 5 1 0	180	1
0 1 5 1 0	106920	176
0 2 5 1 0	55440	104
0 3 5 1 0	10260	23
0 4 5 1 0	900	3
0 5 5 1 0	180	1
0 0 6 1 0	120120	220
1 0 6 1 0	30	1
0 1 6 1 0	124740	212
0 2 6 1 0	35280	73
0 3 6 1 0	2700	10
0 4 6 1 0	150	2
0 5 6 1 0	30	1
0 0 7 1 0	171600	297
0 1 7 1 0	106920	176
0 2 7 1 0	14400	33
0 3 7 1 0	300	2

$a_1a_2a_3a_4a_5$	D	R
0 0 8 1 0	193050	334
0 1 8 1 0	66825	118
0 2 8 1 0	3420	11
0 0 9 1 0	171600	297
0 1 9 1 0	29700	57
0 2 9 1 0	360	3
0 0 10 1 0	120120	220
0 1 10 1 0	8910	21
0 0 11 1 0	65520	124
0 1 11 1 0	1620	5
0 0 12 1 0	27300	62
0 1 12 1 0	135	2
0 0 13 1 0	8400	23
0 0 14 1 0	1800	8
0 0 15 1 0	240	2
0 0 16 1 0	15	1
0 0 0 2 0	105	2
1 0 0 2 0	60	1
0 1 0 2 0	540	4
1 1 0 2 0	60	1
0 2 0 2 0	1170	7
0 3 0 2 0	1380	7
0 4 0 2 0	945	6
0 5 0 2 0	360	3
0 6 0 2 0	60	1
0 0 1 2 0	1320	6
1 0 1 2 0	180	1
0 1 1 2 0	4680	13
0 2 1 2 0	6480	17
0 3 1 2 0	4380	13
0 4 1 2 0	1440	5
0 5 1 2 0	180	1
0 0 2 2 0	7650	23
1 0 2 2 0	180	1
0 1 2 2 0	18000	40
0 2 2 2 0	14940	36
0 3 2 2 0	5220	15
0 4 2 2 0	900	3
0 5 2 2 0	180	1
0 0 3 2 0	27060	61
1 0 3 2 0	60	1
0 1 3 2 0	40320	80
0 2 3 2 0	18360	43
0 3 3 2 0	2820	10
0 4 3 2 0	300	2
0 5 3 2 0	60	1
0 0 4 2 0	65175	133
0 1 4 2 0	57960	109
0 2 4 2 0	12690	30
0 3 4 2 0	600	3
0 0 5 2 0	112860	208
0 1 5 2 0	55440	104

$a_1a_2a_3a_4a_5$	D	R
0 2 5 2 0	4680	14
0 0 6 2 0	144540	266
0 1 6 2 0	35280	73
0 2 6 2 0	720	4
0 0 7 2 0	138600	252
0 1 7 2 0	14400	33
0 0 8 2 0	99495	191
0 1 8 2 0	3420	11
0 0 9 2 0	52800	108
0 1 9 2 0	360	3
0 0 10 2 0	20130	50
0 0 11 2 0	5220	16
0 0 12 2 0	825	5
0 0 13 2 0	60	1
0 0 0 3 0	455	5
1 0 0 3 0	60	1
0 1 0 3 0	1260	6
0 2 0 3 0	1380	7
0 3 0 3 0	800	5
0 4 0 3 0	300	2
0 5 0 3 0	60	1
0 0 1 3 0	4400	15
1 0 1 3 0	60	1
0 1 1 3 0	7500	18
0 2 1 3 0	4380	13
0 3 1 3 0	1200	6
0 4 1 3 0	300	2
0 5 1 3 0	60	1
0 0 2 3 0	19200	48
0 1 2 3 0	18900	40
0 2 2 3 0	5220	15
0 3 2 3 0	600	3
0 0 3 3 0	49860	102
0 1 3 3 0	26100	52
0 2 3 3 0	2820	10
0 0 4 3 0	85530	169
0 1 4 3 0	21300	46
0 2 4 3 0	600	3
0 0 5 3 0	101640	190
0 1 5 3 0	10260	23
0 0 6 3 0	85260	167
0 1 6 3 0	2700	10
0 0 7 3 0	50400	106
0 1 7 3 0	300	2
0 0 8 3 0	20535	51
0 0 9 3 0	5480	18
0 0 10 3 0	860	6
0 0 11 3 0	60	1

Table B.2: $D(6)$ and $R(6)$ by profile (continued).

$a_1a_2a_3a_4a_5$	D	R
0 0 0 4 0	1365	9
1 0 0 4 0	30	1
0 1 0 4 0	1890	9
0 2 0 4 0	945	6
0 3 0 4 0	300	2
0 4 0 4 0	150	2
0 5 0 4 0	30	1
0 0 1 4 0	9900	26
0 1 1 4 0	7200	18
0 2 1 4 0	1440	5
0 3 1 4 0	300	2
0 0 2 4 0	31500	72
0 1 2 4 0	10800	25
0 2 2 4 0	900	3
0 0 3 4 0	57660	115
0 1 3 4 0	8160	22
0 2 3 4 0	300	2
0 0 4 4 0	66900	137
0 1 4 4 0	3420	10
0 0 5 4 0	51120	103
0 1 5 4 0	900	3
0 0 6 4 0	25950	63
0 1 6 4 0	150	2
0 0 7 4 0	8700	25
0 0 8 4 0	1935	8
0 0 9 4 0	300	2
0 0 10 4 0	30	1
0 0 0 5 0	3003	15
1 0 0 5 0	6	1
0 1 0 5 0	1890	9
0 2 0 5 0	360	3
0 3 0 5 0	60	1
0 4 0 5 0	30	1
0 5 0 5 0	6	1
0 0 1 5 0	15840	38
0 1 1 5 0	4140	11
0 2 1 5 0	180	1
0 3 1 5 0	60	1
0 0 2 5 0	35280	80
0 1 2 5 0	3150	10
0 2 2 5 0	180	1
0 0 3 5 0	43140	91
0 1 3 5 0	1200	6
0 2 3 5 0	60	1
0 0 4 5 0	31605	74
0 1 4 5 0	450	2
0 0 5 5 0	14364	38
0 1 5 5 0	180	1
0 0 6 5 0	4260	16
0 1 6 5 0	30	1
0 0 7 5 0	1020	6

$a_1a_2a_3a_4a_5$	D	R
0 0 8 5 0	270	2
0 0 9 5 0	60	1
0 0 10 5 0	6	1
0 0 0 6 0	5005	21
0 1 0 6 0	1260	6
0 2 0 6 0	60	1
0 0 1 6 0	18480	44
0 1 1 6 0	1320	5
0 0 2 6 0	27300	64
0 1 2 6 0	360	1
0 0 3 6 0	20700	48
0 1 3 6 0	120	1
0 0 4 6 0	8745	28
0 0 5 6 0	2220	8
0 0 6 6 0	420	2
0 0 7 6 0	60	1
0 0 0 7 0	6435	24
0 1 0 7 0	540	4
0 0 1 7 0	15840	38
0 1 1 7 0	180	1
0 0 2 7 0	14400	38
0 0 3 7 0	6060	19
0 0 4 7 0	1350	7
0 0 5 7 0	180	1
0 0 0 8 0	6435	24
0 1 0 8 0	135	2
0 0 1 8 0	9900	26
0 0 2 8 0	4950	17
0 0 3 8 0	960	5
0 0 4 8 0	135	2
0 0 0 9 0	5005	21
0 1 0 9 0	15	1
0 0 1 9 0	4400	15
0 0 2 9 0	1000	6
0 0 3 9 0	60	1
0 0 4 9 0	15	1
0 0 0 10 0	3003	15
0 0 1 10 0	1320	6
0 0 2 10 0	90	1
0 0 0 11 0	1365	9
0 0 1 11 0	240	2
0 0 0 12 0	455	5
0 0 1 12 0	20	1
0 0 0 13 0	105	2
0 0 0 14 0	15	1
0 0 0 15 0	1	1
0 0 0 0 1	6	1
1 0 0 0 1	6	1
0 1 0 0 1	30	1
0 2 0 0 1	60	1
0 3 0 0 1	60	1

$a_1a_2a_3a_4a_5$	D	R
0 4 0 0 1	30	1
0 5 0 0 1	6	1
0 0 1 0 1	60	1
0 1 1 0 1	180	1
0 2 1 0 1	180	1
0 3 1 0 1	60	1
0 0 2 0 1	270	2
0 1 2 0 1	450	2
0 2 2 0 1	180	1
0 0 3 0 1	720	4
0 1 3 0 1	600	3
0 2 3 0 1	60	1
0 0 4 0 1	1260	6
0 1 4 0 1	450	2
0 0 5 0 1	1512	6
0 1 5 0 1	180	1
0 0 6 0 1	1260	6
0 1 6 0 1	30	1
0 0 7 0 1	720	4
0 0 8 0 1	270	2
0 0 9 0 1	60	1
0 0 10 0 1	6	1
0 0 0 1 1	60	1
0 1 0 1 1	120	1
0 2 0 1 1	60	1
0 0 1 1 1	420	2
0 1 1 1 1	360	1
0 0 2 1 1	1260	4
0 1 2 1 1	360	1
0 0 3 1 1	2100	6
0 1 3 1 1	120	1
0 0 4 1 1	2100	6
0 0 5 1 1	1260	4
0 0 6 1 1	420	2
0 0 7 1 1	60	1
0 0 0 2 1	270	2
0 1 0 2 1	180	1
0 0 1 2 1	1260	4
0 1 1 2 1	180	1
0 0 2 2 1	2340	7
0 0 3 2 1	2160	6
0 0 4 2 1	990	4
0 0 5 2 1	180	1
0 0 0 3 1	720	4
0 1 0 3 1	120	1
0 0 1 3 1	2100	6
0 0 2 3 1	2160	6
0 0 3 3 1	900	4
0 0 4 3 1	120	1
0 0 0 4 1	1260	6

$a_1a_2a_3a_4a_5$	D	R
0 1 0 4 1	30	1
0 0 1 4 1	2100	6
0 0 2 4 1	990	4
0 0 3 4 1	120	1
0 0 4 4 1	30	1
0 0 0 5 1	1512	6
0 0 1 5 1	1260	4
0 0 2 5 1	180	1
0 0 0 6 1	1260	6
0 0 1 6 1	420	2
0 0 0 7 1	720	4
0 0 1 7 1	60	1
0 0 0 8 1	270	2
0 0 0 9 1	60	1
0 0 0 10 1	6	1
0 0 0 0 2	15	1
0 1 0 0 2	15	1
0 0 1 0 2	60	1
0 0 2 0 2	90	1
0 0 3 0 2	60	1
0 0 4 0 2	15	1
0 0 0 1 2	90	1
0 0 1 1 2	180	1
0 0 2 1 2	90	1
0 0 0 2 2	225	2
0 0 1 2 2	180	1
0 0 0 3 2	300	3
0 0 1 3 2	60	1
0 0 0 4 2	225	2
0 0 0 5 2	90	1
0 0 0 6 2	15	1
0 0 0 0 3	20	1
0 0 1 0 3	20	1
0 0 0 1 3	60	1
0 0 0 2 3	60	1
0 0 0 3 3	20	1
0 0 0 0 4	15	1
0 0 0 1 4	15	1
0 0 0 0 5	6	1
0 0 0 0 6	1	1

Table B.3: $D(6)$ and $R(6)$ by profile (continued).