Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Slicing the n-cube

Practical Work

M. Nowack

January 21, 2022

Advisors: Prof. Dr. B. Gärtner, S. Ilchi

Department of Computer Science, ETH Zürich

Abstract

We re-implement in modern C++ an old algorithm that was used to computationally assert that the minimum number of hyperplanes needed to slice all the edges of the 5-dimensional Boolean hypercube is 5. An edge is sliced by a hyperplane if its two endpoints lie on different sides of the hyperplane. Through algorithmic and programming improvements we reduce the previous running time by a factor of over one thousand. We extend this framework to computationally study other aspects of slicing the n-dimensional Boolean hypercube such as what happens if the normal vector of the hyperplanes contains only small integer weights or generalizing hyperplanes to n-variate polynomials.

Contents

Chapter 1

Introduction

Consider an n-variate polynomial $p : \mathbb{R}^n \to \mathbb{R}$. It decomposes the n-dimensional space into two regions. One, region P, is the set of point that is evaluated positive on p, and the other region, let's call it N, is the set of points that make p negative. In other words, the n-dimensional hypersurface $p = 0$ *slices* $\mathbb{R}^n$ into P and N and there is an extensive study in algebraic geometry to describe the geometry of this slice.

In computer science, same question arises but with the focus on studying sign of p on the n-dimensional Boolean hypercube, in short n-cube. As n-cube plays the key role in our discussion, let us define it here.

Definition 1.1 *For any positive integer n, an n-cube is the convex hull of the points in $\{-1, +1\}^n$.*

Here are some basic observations on the n-cube:

- It has 2^n vertices and $2^{n-1}n$ edges.
- There is an edge between $x = (x_1, \dots, x_n) \in \{-1, +1\}^n$ and $y = (y_1, \dots, y_n) \in \{-1, +1\}^n$ if there is an $i \in [n]$ such that $x_i \neq y_i$ and for all $j \in [n] \setminus \{i\}$, $x_j = y_j$.

Assuming that p is not zero on the vertices of n-cube, we now redefine the set P (respectively, N) as the set of vertices that has positive (negative) value on p. Here are some applications in which researchers trying to understand P and N better:

- **Learning Theory**: Consider a family of Boolean functions

  $$\mathcal{F} = \{f : \{-1, +1\}^n \to \{-1, +1\} \mid f \text{ has some suitable structure}\}$$

  and a hidden function $f^* \in \mathcal{F}$. We want to find f^* or at least an approximation of it, given S, a set of m of sample points. Each sample point is a pair (x, ℓ) where $x \in \{-1, +1\}^n$ and $\ell = f^*(x)$. The question now is how many samples do we need for recovery of f^*? Of course, this depends on the complexity of $\mathcal{F}$, i.e. how restrictive the "suitable structure" in the definition of $\mathcal{F}$ is. One way to capture this complexity is to compute how well $\mathcal{F}$ can be represented by low-degree polynomials. Note that each $f \in \mathcal{F}$ is essentially a bi-partition of $\{-1, +1\}^n$. We can think of them as sets P and N in our definition. This line of works has been started in the work of Minsky and Paper 1968 [9] for studying Perceptron and it is still an ongoing research topic [8, 3, 4].

- **Computational Complexity**: One major research topic in Computational Complexity is to prove lower bound for the amount of computational resources (e.g. time, space, randomization) we need to evaluate a Boolean function. For example, we know that PARITY function, i.e. $f(x_1, \ldots, x_n) = \prod_{i=1}^n x_i$ cannot be computed with polynomially many AND, OR, and NOT gates in logarithmic depth (circuit depth is a measure of time in the parallel setting). What if we have more complex gates like the gates that can compute the sign of a low-degree polynomial function? What about other families, like DNFs? Moreover, like the point that we mention above, measuring the complexity of a family of functions with polynomials has been received a lot of attention in this areas [13, 7, 15].

1.1 Slicing n-cube with hyperplanes

Our focus is to study the following question:

> **Question:** How many degree-1 polynomials are needed to slice all the edges of the n-cube? An edge (x, y) is sliced with polynomial p if $\text{sgn}(p(x)) \neq \text{sgn}(p(y))$.

Since $p = 0$ is a hyperplane, through the thesis, we may also refer to hyperplanes as the slicing object. The question above has an easy upper bound of n. Consider n hyperplanes $\{x_i = 0\}_{i=1}^n$. In 1971, O'Neil [10] proved that any hyperplane that can cut at most $\frac{1}{\sqrt{n}}$ fraction of the edges. This directly implies a lower bound of $\Omega(\sqrt{n})$ for our objective. The lower bound has been recently improved to $\Omega(n^{0.51})$ by Yehuda and Yehudayoff [16].

On the upper bound side, the only construction that needs less than n hyperplanes is based on the unpublished result of M. Paterson [11]. He showed that for $n = 6$, only 5 hyperplanes are needed. This automatically implies that for any n, $\lceil \frac{5n}{6} \rceil$ many hyperplanes are enough. Here is a quick proof for that. Consider $(u^{(1)}, t_1), \ldots, (u^{(5)}, t_5) \in \mathbb{R}^6 \times \mathbb{R}$ where $(u^{(i)}, t_i)$ represents the i-th hyperplane in Paterson's construction, i.e. $\{w \in \mathbb{R}^6 \mid w^T u^{(i)} = t_i\}$. Now consider an edge of the n-cube between vertex x and y and suppose that x and y disagree at coordinate i. We extend $u^{(1)}$ to n

dimensions as follows. First, let $r = \lfloor \frac{i-1}{6} \rfloor$, so $6r + 1 \le i \le 6(r+1)$. Define $v^{(1)} \in \mathbb{R}^n$ such that $v^{(1)}_{6r+j} = u^{(1)}_j$ for $j \in [6]$. All the other coordinates are $v^{(1)}$ are zero. Similarly, we define $v^{(2)}, \dots, v^{(5)}$. Since $\{u^{(i)}\}_{i\in[n]}$ slices 6-cube, it is easy to see that at least one of $v^{(i)}$ should slice the edge (x, y). From this, and by decomposing the n coordinates into $\lceil \frac{n}{6} \rceil$ groups of size at most 6, we can slice all the edge of n-cube with $5\lceil \frac{n}{6} \rceil$ many hyperplanes.

1.2 Our work

Let $f_{\text{SC}}(n)$ be the smallest number of hyperplanes needed to slice all the edges of the n-cube. From the discussion above, we know that $f_{\text{SC}}(n) \le 5\lceil \frac{n}{6} \rceil$ and $f_{\text{SC}}(n) = \Omega(n^{0.51})$. In this thesis, we focus on the upper bound. Like extending the construction of Paterson, one way to attack the upper bound is through considering the exact behaviour of $f_{\text{SC}}(n)$ for small n.

For $n = 2$ and $n = 3$, it is easy to show that $f_{\text{SC}}(n) = n$. For $n = 4$, Emamy, shows that $f_{\text{SC}}(4) = 4$. In his next work with Arca-Nazario, they show that even for $n = 5$, we have $f_{\text{SC}(5)} = 5$. From the Paterson's construction we know that $f_{\text{SC}}(6) \le 5$. It is also easy to see that $f_{\text{SC}}(n) \le f_{\text{SC}}(n+1)$. So $f_{\text{SC}}(6) = 5$.

Prior to the theoretical proof of $f_{\text{SC}}(5) = 5$, Sohler and Ziegler [14] used a computer program to computationally assert this claim. We re-implement in modern C++ this algorithm and through algorithmic and programming improvements we reduce the previous running time by a factor of over one thousand. We then extend this framework to computationally study other aspects of slicing the n-cube:

- We present a detailed breakdown for $n \le 7$ of how many hyperplanes there are that slice as many edges as possible and how many edges each such hyperplane slices.
- We determine upper bounds on the minimum number of degree-2 polynomials needed to slice all the edges of the n-cube.
  - For $n = 2$, at most two degree-2 polynomials are needed.
  - For $n = 3$, at most two degree-2 polynomials are needed.
  - For $n = 4$, at most two degree-2 polynomials are needed.
  - For $n = 5$, at most three degree-2 polynomials are needed.
- We find that if all normal vectors have coefficients ± 1, the minimum number of hyperplanes needed to slice all the edges of the n-cube is n for $n \le 7$. This still holds if the threshold is additionally restricted to $\{0, 1\}$.

- We find the smallest m so that the hyperplanes whose normal vector has only integer coefficients $\{-m, \dots, m\}$ are equivalent to the hyperplanes whose normal vector contains arbitrary real values for the purpose of slicing edges of the n-cube.
  - For $n = 2$, the smallest m is 1.
  - For $n = 3$, the smallest m is 2.
  - For $n = 4$, the smallest m is 3.
  - For $n = 5$, the smallest m is 5.
  - For $n = 6$, the smallest m is at least 7 on the basis of intermediate results.
- We argue why it is computationally infeasible to determine $f_{\text{SC}}(7)$ by this algorithm despite its improvements.

1.3 Outline of this thesis

In Chapter 2, we review several theoretical properties of slicing n-cube edges with hyperplanes, e.g. maximum number of edges that can be sliced with one hyperplane and how many different ways we can slice the edges with one hyperplane. In Chapter 3, we present the algorithm that we implemented. In the final chapter 4, we elaborate on the set of results we found using a computer program.

Chapter 2

Survey on basic theoretical properties

In this chapter, we review some of the basic properties of slicing the n-cube with hyperplanes from the literature. First, we show that each hyperplane can slice at most $\sqrt{\frac{1}{n}}$ fraction of edges of the n-cube. This gives us a lower bound of $\sqrt{n}$ on the number of hyperplanes needed for slicing all the edges. Next, we push this further and show that even intersection of k-hyperplanes (bounded and unbounded polytopes) can slice at most $\sqrt{\frac{\log k}{n}}$ fraction of edges. In the last part, we show that there are $(n!)^{O(n)}$ many ways to slice the edges of the n-cube with hyperplanes. For this, we prove if S is a subset of edges that can be sliced with one hyperplane, there should be a hyperplane for which all of the coordinates of its normal vector is bounded by $(n!)^{O(1)}$ and it slices exactly S.

2.1 Slicing maximum number of edges with a hyperplane

In this section, we want to prove that the maximum number of edges that can be sliced with one hyperplane is $O(2^n\sqrt{n})$. Since the total number of edges in a hypercube is $O(2^n n)$, this directly implies a $\sqrt{n}$ bound on the maximum possible. The proof is based on the work by O'Neil [10].

Essentially, we are looking for a normal vector $u \in \mathbb{R}^n$ and a threshold $t \in \mathbb{R}$ such that the number of edges sliced by $\{x \in \mathbb{R}^n \mid u^T x = t\}$ is maximized. Without loss of generality, we can assume that all the coefficients of u are positive. There is a simple way to achieve slicing $O(2^n\sqrt{n})$. Just consider the majority function. More concretely, let $u = (1, \ldots, 1)^T$ be the all ones vector and $t = 0$. For simplicity, assume n is odd. Now, an edge $(x, y) \in \{-1, +1\}^n \times \{-1, +1\}^n$ is sliced by the hyperplane (u, t) if and only if $u^T x = -1$ and $u^T y = +1$ or vice versa. In other words, all the edges between vertices with $\frac{n-1}{2}$ ones in their coordinate and vertices with $\frac{n+1}{2}$ ones in their coordinate are sliced. This number is equal to

$$\binom{n}{\frac{n-1}{2}}\frac{n+1}{2} = O(\frac{2^n}{\sqrt{n}}) \cdot \frac{n+1}{2} = O(2^n\sqrt{n}). \tag{2.1}$$

The first equality comes from Stirling's approximation. Next, we want to prove that for any u and t, this is the best possible up to some constant factor. For that, we need the following partial ordering over the vertices and edges of the n-cube.

Definition 2.1 *We say a vertex $x \in \{-1,+1\}^n$ is less than vertex $y \in \{-1,+1\}^n$, if $x \neq y$ and for all $i \in [n]$, we have $x_i \leq y_i$.*

Note that the endpoints of each edge is comparable from this definition as they only disagree on one coordinate. From now on, whenever we represent an edge $e = (x,y) \in \{-1,+1\}^n \times \{-1,+1\}^n$, we assume that $x < y$.

Definition 2.2 *Edge $e = (x,y)$ is less than edge $e' = (x',y')$ if $y < x'$. In words, e is less than e' if the maximum of the two endpoints of e is less than the minimum of the two endpoints of e'.*

Lemma 2.3 *Assuming that all the coefficients of u are positive and it slices edges $e = (x,y)$ and $e' = (x',y')$, then e and e' are non-comparable on the ordering above.*

Proof Suppose this is not the case and without loss of generality assume $e < e'$. First, note that since u has positive coefficients, we have

$$u^T x < t, u^T y > t, \quad u^T x' < t, u^T y' > t.$$

Let $\delta x = x' - y$. Since $e < e'$, so $y < x'$ and as a result all the coordinates of δx is non-negative. In particular, each coordinate is either zero or two. So we have

$$t < u^T y \leq u^T (y + \delta x) = u^T x'.$$

This contradicts the fact that $u^T x' < t$. □

From this lemma, it is enough if we can bound the maximum number of non-comparable edges. As you may know, from the famous Sperner's theorem, the maximum number of non-comparable vertices is $O(2^n\sqrt{\frac{1}{n}})$, or more precisely, it is $\binom{n}{\frac{n-1}{2}}$. There is a generalization of Sperner's theorem, proved by Baker [2], which shows that the maximum number of non-comparable edges is $O(2^n\sqrt{n})$. This theorem also gives a sharp bound of $\binom{n}{\frac{n-1}{2}}\frac{n+1}{2}$ which can be achieved by $u = (1,\dots,1)$, and $t = 0$.

2.2 Average sensitivity for intersection of hyperplanes

As we have discussed in the introduction, measuring the complexity of a class of functions with polynomials has been studied extensively in literature. Here, we want to discuss one of these results in relation with hyperplanes. Consider an n-variate Boolean function $f : \{-1, +1\}^n \to \{-1, +1\}$ over the n-cube. One common way to define complexity of a function is via its average sensitivity.

Definition 2.4 *For a function $f : \{-1, +1\}^n \to \{-1, +1\}$, the average sensitivity is defined as*

$$\mathrm{AS}(f) = \mathbb{E}_{x \in \{-1,+1\}^n}[|\{y \in \{-1, +1\}^n \mid (x, y) \textit{ is an edge}\}|]$$

where x is a uniformly sampled vertex of n-cube.

Observe that if f is the corresponding function of hyperplane h, then the average sensitivity of f is just a normalized version (a division by factor 2^{n-1}) of the number of edges slices by h.

In this section, we want to prove that the set of functions that can be defined as the intersection of k hyperplanes has average sensitivity $O(\sqrt{n \log k})$. Note that for $k = 1$, we already prove this in the previous section. The result of this part is based on the work of Kane [7].

For k hyperplanes $(u_1, t_1), \ldots, (u_k, t_k)$ where the i-th hyperplane is $\{x \in \mathbb{R}^n \mid u_i^T x = t_i\}$, a function $f : \{-1, +1\}^n \to \{-1, +1\}$ is defined as follows:

$$\forall x \in \{-1, +1\}^n, \quad f(x) = 1 \iff u_i^T x > t_i \text{ for all } i \in [k]$$

We call f is the intersection of the k hyperplanes. In this section, we want to prove the following theorem.

Theorem 2.5 *For any $f : \{-1, +1\}^n \to \{-1, +1\}$ which is the intersection of k hyperplanes, $\mathrm{AS}(f) = O(\sqrt{n \log k})$.*

For this, first we define unate functions.

Definition 2.6 *A function $f : \{-1, +1\}^n \to \{-1, +1\}$ is unate if for all $i \in [n]$, function f is either increasing with respect to the i-th coordinate or decreasing with respect to the i-th coordinate. In other words, if $x, y \in \{-1, +1\}^n \to \{-1, +1\}$ such that $x_i = -1$ and $y_i = +1$, and they agree on all the other coordinates, then it is always either $f(x) \leq f(y)$ or $f(x) \geq f(y)$ regardless of x.*

Theorem 2.7 *If $f_1, \ldots, f_k$ are unate functions, then $F = \max\{f_i\}_{i \in [k]}$ has average sensitivity of $O(\sqrt{n \log k})$.*

Proof (Theorem 2.5) A function with respect to a hyperplane (u, t) is unate. More concretely, it is increasing with respect to the i-th coordinate if u_i is non-negative and its decreasing otherwise. Observe that that intersection of k hyperplanes, $f_1, \dots, f_k$, their intersection is defined by $F = -\max\{f_i\}_{i\in[k]}$. Since $\text{AS}(F) = \text{AS}(-F)$, the claim follows form Theorem 2.7. $\square$

Proof (Theorem 2.7) To prove the claim, we need the following auxiliary lemma.

Lemma 2.8 *For any Boolean function $f : \{-1, +1\}^n \to \{-1, +1\}$, and $\mu = \mathbb{E}_{x\sim\{-1,+1\}^n}[\frac{f(x)+1}{2}]$, we have*

$$\mathbb{E}_{x\sim\{-1,+1\}^n}\left[(f(x)+1)/2\sum_{i\in[n]} x_i\right] = O(\mu\sqrt{n\log\mu^{-1}}).$$

Proof For ease of notation, let $Z(x) = (f(x)+1)/2\sum_{i\in[n]} x_i$. The claim can be proved in several steps of calculations:

$$\begin{aligned}
\mathbb{E}_{x\sim\{-1,+1\}^n}[Z(x)] &\le \int_0^\infty \mathbb{P}[Z(x)\ge\theta]d\theta && (2.2)\\
&\le \int_0^\infty \min(\mu, \mathbb{P}[\sum_{i\in[n]} x_i \ge \theta])d\theta && (2.3)\\
&\le \int_0^\infty \min(\mu, e^{-\Omega(\frac{\theta^2}{n})})d\theta && (2.4)\\
&\le \int_0^{\sqrt{n\log\mu^{-1}}} p d\theta + \int_{\sqrt{n\log\mu^{-1}}}^\infty e^{-\Omega(\frac{\theta^2}{n})}d\theta && (2.5)\\
&= O(\mu\sqrt{n\log\mu^{-1}}). && (2.6)
\end{aligned}$$

Equation (2.2) comes from basic definition of expectation. For equation (2.3), note that $\frac{f(x)+1}{2} \in \{0, 1\}$. So μ is the probability that $\frac{f(x)+1}{2}$ equals 1. Since θ is positive, $Z(x)$ can be only greater that or equal to θ if $\frac{f(x)+1}{2}$ is 1. Equation (2.4) comes from the Chernoff bound on the sum of n Bernoulli random variables. Equations (2.5) and (2.6) uses basic rules for computing integrals. $\square$

To compute $\text{AS}(F)$, we use the sequence of functions $F_i = \max_{j\in[i]} f_i$. So $F = F_k$. Define $G_i = F_i - F_{i-1} - 1$. Note that $F_i(x) \ge F_{i-1}(x)$ for all x, so G_i is a function from $\{-1, +1\}^n$ to $\{-1, +1\}$. Let $\mu_i = \mathbb{E}_{x\sim\{-1,+1\}^n}[G_i(x)]$. Our goal is to show

$$\text{AS}(F_i) - \text{AS}(F_{i-1}) = O(\mu_i\sqrt{n\log\mu_i^{-1}})$$

If this claim is correct, the final bound follows easily. Note that:

$$\begin{aligned}\mathrm{AS(F)} &= \sum_{i\in[k]} \mathrm{AS}(F_i) - \mathrm{AS}(F_{i-1}) \\ &= O(\sum_{i\in[k]} \sqrt{n}\mu_i\sqrt{\log \mu_i^{-1}}).\end{aligned}$$

Function $q : \mu \to \mu_i\sqrt{\log \mu_i^{-1}}$ is concave for $\mu \in (0,1)$. Each μ_i is also in this range. So for $M = \frac{\sum_{i\in[k]}\mu_i}{k}$, we have $\sum_{i\in[k]} \mu_i\sqrt{\log \mu_i^{-1}} \leq kM\sqrt{\log M^{-1}}$. Replacing this in the equation above, we get

$$\mathrm{AS}(F) = O(M\sqrt{n\log kM^{-1}}) = O(\sqrt{n\log kM}) = O(\sqrt{n\log k}).$$

So the only remaining piece is to prove that $\mathrm{AS}(F_i) - \mathrm{AS}(F_{i-1}) = O(\mu_i\sqrt{n\log \mu_i^{-1}})$ for all i. Recall that f_i is a unate function. Without loss of generality, we assume that it is increasing on each coordinate. We show that

$$\mathrm{AS}(F_i) - \mathrm{AS}(F_{i-1}) = \sum_{j=1}^{n} \mathbb{E}_x[|F_i(x) - F_i(x^{(j)}| - |F_{i-1}(x) - F_{i-1}(x^{(j)}|]$$

where $x^{(j)}$ is same as x in all the coordinates expect that its j-th coordinate is negated. To prove this, we prove the following auxiliary lemma.

Lemma 2.9 *For any $x \in \{-1,+1\}^n$ and $j \in [n]$, we have:*

$$|F_i(x) - F_i(x^{(j)}| - |F_{i-1}(x) - F_{i-1}(x^{(j)}| \leq x_i((F_i(x) - F_i(x^{(j)}) - (F_{i-1}(x) - F_{i-1}(x^{(j)})).$$

Proof If $f_i(x) = f_i(x^j) = 0$, then $F_i(x) = F_{i-1}(x)$ and $F_i(x^{(j)}) = F_{i-1}(x^{(j)})$ and the claim is trivial. Suppose $f_i(x) = 1$ and $x_j = 1$. Recall that we assume f_i is increasing on all the coordinates, so with this assumption we have $1 = f_i(x) \geq f_i(x^{(j)})$. The claim follows by simple calculations. The case where $f_i(x^j) = 1$ is also similar. □

Now, let's get back to bounding $\mathrm{AS}(F_i) - \mathrm{AS}(F_{i-1})$. From what we have seen, we can derive the following:

$$
\begin{aligned}
\mathrm{AS}(F_i) - \mathrm{AS}(F_{i-1}) &= \sum_{j=1}^{n} \mathbb{E}_x[|F_i(x) - F_i(x^{(j)}| - |F_{i-1}(x) - F_{i-1}(x^{(j)}|] \\
&\leq \sum_{j=1}^{n} \mathbb{E}_x[x_i((F_i(x) - F_i(x^{(j)}) - (F_{i-1}(x) - F_{i-1}(x^{(j)}))] \\
&= \sum_{j=1}^{n} \mathbb{E}_x[x_j(G_i(x) - G_i(x^{(j)}))] \\
&= \sum_{j=1}^{n} \mathbb{E}_x[x_j G_i(x)] - \sum_{j=1}^{n}[-x_j^{(j)} G_i(x^{(j)})] \\
&= 2\mathbb{E}_x[G_i(x) \sum_{j=1}^{n} x_j] \\
&= O(\mu_i \sqrt{n \log \mu_i^{-1}})
\end{aligned}
$$

This completes the proof. □

2.3 Integer representation of hyperplanes

Suppose we are only allowed to have hyperplanes with integer coefficients. How large the coefficients need to be to realize all the possible hyperplanes? In this section we show that magnitude $(n!)^{O(1)}$ is enough. This is based on the proof of Hastad [6].

Let $h = (u, t)$ be an arbitrary hyperplane and let $f : \{-1, +1\}^n \to \{-1, +1\}$ be its corresponding function. By some small perturbation on the coordinates of u, we can assume that for all $x \in \{-1, +1\}^n$, the value $u^T x$ is not zero. Now scale coordinates of u and t such that $\min_{x \in \{-1,+1\}^n} |u^T x| \geq 1$. Let $h^* = (u^*, t^*)$ as the hyperplane that realizes exactly the function corresponding to h, such that we have also $\min_{x \in \{-1,+1\}^n} |u^{*T} x| \geq 1$. Moreover, it's the one that maximizes the number of point x on the n-cube such that $|u^{*T} x|$ exactly equals to 1.

We want to use u^* to find the hyperplane with bounded integer coordinates on its normal vector that realizes h. Let $x^{(1)}, \ldots, x^{(k)} \in \{-1, +1\}^n$ be the set of points such that $|u^{*T} x^{(i)}| = 1$.

Lemma 2.10 *The hyperplane h^* can be determined as the solution of the following linear system on the variable u^**

$$u^{*T} x^{(i)} = f(x^{(i)}), \text{ for all } i \in [k]$$

Proof We know this system has a solution, u^*. Suppose it is not the only one. So there should be a non-zero $v \in \mathbb{R}^n$ such that $u^* + \alpha v$ is also a solution for all α. Since v is not zero, there should be an x such that $v^T x \neq 0$. Without loss of generality, assume $v^T x > 0$. Moreover, $x \notin \{x^{(1)}, \ldots, x^{(k)}\}$ since we know $v^T x^{(i)} = 0$. The other fact is that $|u^{*T} x| > 1$ because of the maximality of u^*. Again, without loss of generality, assume $u^{*T} x < -1$.

Let $\alpha > 0$ be the minimum of all the rational α such that $u^* + \alpha v$ has magnitude 1 on of the vertices of n-cube. There should be such an α because of x and observe that $u^* + \alpha v$ realizes f which contradicts the fact that u^* is maximal. □

From the lemma above, u^* can be determined by solving the above linear system on n variables, coordinates of u^*. Moreover, the RHS of each linear equality is either -1 or $+1$. We can solve this system with Cramer's rules. First, we define $(n+1) \times (n+1)$ matrix based on LHS and RHS of the equations. All the entries of this matrix is $+1$ and -1 and we know that the determinant of this matrix is a some of $n!$ terms. So the terms involved in the final solution are rations numbers with numerator and denominator $(n!)^{O(1)}$ which concludes in the following theorem.

Theorem 2.11 *For any hyperplane h, its corresponding function $f : \{-1, +1\}^n \to \{-1, +1\}$ can be realized by a hyperplane where all the coefficients of its normal vector is $(n!)^{O(1)}$.*

Corollary 2.12 *Number of different functions that can be represented by hyperplanes is $(n!)^{O(n)}$.*

Proof From Theorem 2.11, we only need to scan the hyperplanes with weights bounded by $(n!)^{O(1)}$. There are n coordinates, so we have $(n!)^{O(n)}$ many relevant hyperplanes. For each fixed hyperplane, there is at most 2^n, number of vertices of the n-cube, many thresholds that results in a different function which concludes the proof. □

Chapter 3

Algorithm

In this chapter we present the algorithm by Sohler and Ziegler [14] that computationally asserted $f_{\text{SC}}(5) = 5$, i.e. 5 hyperplanes are needed to slice all the edges of the 5-cube. Their program asserts no 4 hyperplanes can slice all edges of the 5-cube and thus $f_{\text{SC}}(5) > 4$. Since $f_{\text{SC}}(n) \leq n$, it follows that $f_{\text{SC}}(5) = 5$.

We first explain the gist of the algorithm and then discuss its three essential subroutines in detail. Let $G_n = (V_n, E_n)$ denote the n-cube from here on.

Definition 3.1 *A set $S \subseteq E_n$ is sliceable if some common hyperplane slices each edge in S. A sliceable set is said to be maximal if adding any new edge from E_n would make the set unsliceable. A covering of S by k sliceable sets is called a k-slicing of S. S is k-sliceable if a k-slicing for it exists.*

The basic idea is to first compute all sliceable sets and then check if any k sliceable sets cover E_n. However, a naive approach is hopelessly inefficient as there are $O(2^{|E|}) = O(2^{n \cdot 2^n})$ many sliceable sets and the number of possible coverings adds another power of k to the complexity.

Fortunately, only coverings made up from maximal sliceable sets (MSSs) need to be considered because for any k-slicing of some $S \subseteq E_n$ there exists a k-slicing of S where each member is maximal. Furthermore, we can make use of the symmetries inherent to the n-cube. Let Σ_n denote the set of the following $2^n \cdot n!$ symmetries inherent to the n-cube:

- Reflections in direction $i \in [n]$, i.e. mappings $u = (u_1, \ldots, u_i, \ldots, u_n) \mapsto (u_1, \ldots, -u_i, \ldots, u_n)$
- Permutations π of dimensions $u \mapsto (u_{\pi(1)}, \ldots, u_{\pi(n)})$.
- Any of the $2^n \cdot n!$ combinations of the above. E.g. clockwise rotation by 90° in the plane is realized by $(x, y) \mapsto (y, -x)$.

Lemma 3.2 *For $\sigma \in \Sigma_n$ and $x, y \in \mathbb{R}^n$, it holds that $\sigma(x) \cdot \sigma(y) = x \cdot y$. In particular, if a hyperplane $H = \{w \in \mathbb{R}^n : w \cdot u = t\}$ slices an edge $e = (x, y) \in E_n$, then $\sigma(H) = \{w \in \mathbb{R}^n : w \cdot \sigma(u) = t\}$ slices edge $\sigma(e) = (\sigma(x), \sigma(y))$. More generally, if $S \subseteq E_n$ is sliceable by means of H, then so is $\sigma(S) = \{\sigma(e) | e \in S\} \subseteq E_n$ by means of $\sigma(H)$.*

Thus, it suffices to consider for each equivalence class $\Sigma_n(S) = \{\sigma(S) : \sigma \in \Sigma_n\}$ only one unique symmetry representative (USR), for example its lexicographically smallest member. The factor gained by symmetries is not as big as $|\Sigma_n|$ because e.g. E_n is the USR for only one MSS, but as we can see later in Table 4.1 it is not too far away from $|\Sigma_n|$.

3.1 Maximal sliceable sets

First, we have to compute the MSSs. To this end, we employ the fact that each MSS is the set of edges exterior to some cut complex [5].

Definition 3.3 *A cut complex is the subgraph of G_n induced by the vertices $V_n \cap H^+$ for some open halfspace H^+.*

The number of cut complexes has an upper bound of $2^{n(n+1)}$. This illustrates the gain achieved by considering MSSs instead of the double exponentially many sliceable sets. Furthermore, we employ the fact that any cut complex of size i emerges from some cut complex of size $i - 1$ by adding an exterior vertex [5]. Finally, the cut complexes P and $V_n \setminus P$ have the same exterior edges and thus only cut complexes up to size 2^{n-1} need to be considered. This offers an efficient way of finding all cut complexes (or rather their USRs) by means of dynamic programming with respect to their size.

Algorithm 1 Cut complexes

```
Ensure: $\mathcal{A}$ is the set of the USRs of all cut complexes having size $\leq 2^{n-1}$
  $\mathcal{A}_1 \leftarrow$ all USRs of cut complexes having size 1
  for $i = 2$ to $2^{n-1}$ do
      for all $P \in \mathcal{A}_{i-1}$ do
          for all $v \in V_n \setminus P$ adjacent to $P$ do
              $P' \leftarrow P \cup \{v\}$
              if $P'$ is a cut complex then
                  $\mathcal{A}_i \leftarrow \mathcal{A}_i \cup \{\text{USR of } P'\}$
              end if
          end for
      end for
  end for
  $\mathcal{A} \leftarrow \bigcup_{1 \leq i \leq n} \mathcal{A}_i$
```

The USRs of the MSSs (USR-MSSs) are then obtained by taking the exterior edges of the returned cut complexes. The USR-MSSs are easily expanded to the general MSSs by applying all symmetry transformations to each USR-MSS. Considering only USRs of cut complexes throughout the algorithm ensures that in each step $\mathcal{A}_i$ remains small.

It remains to show an efficient way of determining whether a vertex set is a cut complex. A vertex set P can be tested for being of the form $P = V_n \cap H^+$ by means of a linear program. The existence of an accordingly oriented halfspace H^+ with normal vector u and threshold t is equivalent to the feasibility of the following linear constraints with respect to $(u,t) \in \mathbb{R}^n \times \mathbb{R}$:

$$a \cdot u + t < 0 \quad \forall a \in P \qquad \wedge \qquad b \cdot u + t > 0 \quad \forall b \in V_n \setminus P$$

However, these are not valid constraints of a linear program because the inequalities are strict rather than weak. To circumvent this issue we introduce a small ε and reformulate the constraints as

$$a \cdot u + t \leq -\varepsilon \quad \forall a \in P \qquad \wedge \qquad b \cdot u + t \geq \varepsilon \quad \forall b \in V_n \setminus P. \tag{3.1}$$

3.2 Maximal 2-sliceable sets

Next, we compute the maximal 2-sliceable sets (M2SSs). Clearly, the M2SSs are among the pairwise unions of all MSSs. Note that $A \cup B$ is in general not maximal 2-sliceable even if A, B are MSSs. By making use of symmetries this step can be reduced to computing the USR-M2SSs which are among the pairwise unions of all MSSs and USR-MSSs, which we will prove shortly in Lemma 3.4.

Algorithm 2 Pairwise unions

```
Require: $\mathcal{A}$ contains all USR-MSSs, $\mathcal{B}$ contains all MSSs
Ensure: $\mathcal{C}$ contains all USR-M2SSs
  $\mathcal{C} \leftarrow \emptyset$
  for all $A \in \mathcal{A}$ do
      for all $B \in \mathcal{B}$ do
          $X \leftarrow$ USR of $A \cup B$
          if $\{C \in \mathcal{C} | C \supseteq X\} = \emptyset$ then          ▷ Skip non-maximal $X$
              $\mathcal{C} \leftarrow \{C \in \mathcal{C} | C \nsubseteq X\}$     ▷ Discard newly non-maximal slicings
              $\mathcal{C} \leftarrow C \cup \{X\}$
          end if
      end for
  end for
```

The USR-M2SSs are then easily expanded to the general M2SSs by applying all symmetry transformations to each USR-M2SS.

Lemma 3.4 *Let $\mathcal{A}$ be the set of all USR-MSSs. Let $\mathcal{B}$ be the set of all MSSs. Let C be a M2SS. There exist $A \in \mathcal{A}$, $B \in \mathcal{B}$ and $\sigma \in \Sigma$ such that $C = \sigma(A \cup B)$. In particular, the set $\mathcal{C}$ returned by Algorithm 2 contains all USR-M2SSs.*

Proof By definition, $C = \tilde{A} \cup \tilde{B}$ for some MSSs $\tilde{A}, \tilde{B}$. Then,

$$C = \tilde{\sigma}^{-1}(\tilde{\sigma}(\tilde{A} \cup \tilde{B})) = \tilde{\sigma}^{-1}(\tilde{\sigma}(\tilde{A}) \cup \tilde{\sigma}(\tilde{B}))$$

for all $\tilde{\sigma} \in \Sigma$. Now, choose $\tilde{\sigma}$ so that $\tilde{\sigma}(\tilde{B})$ is the USR. Then, taking $\sigma := \tilde{\sigma}^{-1} \in \Sigma$, $A := \tilde{\sigma}(\tilde{A}) \in \mathcal{A}$, and $B := \tilde{\sigma}(\tilde{B}) \in \mathcal{B}$ proves the first claim. For the second claim notice that, as invariant to the algorithm's loop, all members of $\mathcal{C}$ are 2-sliceable. Thus, an USR for each M2SS becomes inserted once. On the other hand, it is skipped or removed only if there is some proper superset in $\mathcal{V}$ which cannot happen for maximal 2-sliceable sets. □

It is important to remark that Algorithm 2 discards non-maximal 2-sliceable sets only on a best effort basis and the set $\mathcal{C}$ may contain additional non-maximal (but valid) 2-sliceable sets. While it always holds that if USR A is a subset of USR B, then all sliceable sets in the equivalence class of A are non-maximal, the reverse implication does not hold in general. If USR A is not a subset of USR B, then A may still be a subset of another sliceable set in the equivalence class of B in which case all sliceable sets in the equivalence class of A are non-maximal. What we can do to obtain exactly the set of all M2SSs and USR-M2SSs is to remove all non-maximal sliceable sets from the expanded $\mathcal{C}$ and subsequently reduce them to their USRs again. However, without a dedicated data structure, that enables efficient subset and superset queries, removing all non-maximal sliceable sets from a list has quadratic running time (with respect to the size of the list). This is computationally infeasible for $n = 5$ as we will see in Table 4.1 because there are over half a billion elements in the list. Thus, it is more practical to proceed to the next step without discarding all non-maximal sliceable sets first.

3.3 4-slicing

Finally, to determine if a 4-slicing of E_5 exists, it suffices to check if any of the pairwise unions of all M2SSs and USR-M2SSs equals E_5. Computing the M4SSs or their USRs is unnecessary. This enables to noticeably reduce the number of pairs that have to be considered because all pairs $A, B \subseteq E_n$ for which $|A| + |B| < |E_n|$ obviously do not slice all edges.

As we will see in Section 4.1, there is actually a different sorting criterion that further reduces the number of pairs that have to be considered by a significant margin. However, this is a novel feature and is thus not discussed in this chapter.

Algorithm 3 Does any pairwise union slice the n-cube?

```
Require: $\mathcal{A}$ contains all USR-M2SSs, $\mathcal{B}$ contains all M2SSs
  Sort $\mathcal{B}$ in descending order with respect to cardinality
  for all $A \in \mathcal{A}$ do
      for all $B \in \mathcal{B}$ do
          if $A \cup B = E_n$ then
              return True
          else if $|A| + |B| < |E_n|$ then
              break
          end if
      end for
  end for
  return False
```

3.4 Slicing the n-cube

The previous sections (Section 3.2 and 3.3) aim at the specific case $n = 5$. Here, we sketch the algorithm to determine the minimum number of hyperplanes needed to slice all the edges of the n-cube for any n using generalized versions of Algorithm 2 and 3 as subroutines. All arguments and proofs work analogously.

Algorithm 4 Minimum number of hyperplanes needed to slice the n-cube

```
Require: USR-MSSs
  for all $A \in$ USR-MSSs do
    if $A = E_n$ then
      return 1
    end if
  end for
  MSSs $\leftarrow \{\sigma(A) | A \in$ USR-MSSs$, \sigma \in \Sigma_n\}$
  if any pairwise union of USR-MSSs and MSSs slices the $n$-cube then
    return 2
  end if
  for $k = 2, 3, \ldots$ do
    $i \leftarrow \lceil k/2 \rceil$
    $j \leftarrow \lfloor k/2 \rfloor$
    USR-M$k$SSs $\leftarrow$ pairwise unions of USR-M$i$SSs and M$j$SSs
    if any pairwise union of USR-M$k$SS and M($k$-1)SS slices the $n$-cube then
      return $2k - 1$
    end if
    M$k$SSs $\leftarrow \{\sigma(A) | A \in$ USR-M$k$SSs$, \sigma \in \Sigma_n\}$
    if any pairwise union of USR-M$k$SSs and M$k$SSs slices the $n$-cube then
      return $2k$
    end if
  end for
```

Chapter 4

Results

All our code including documentation is available at https://gitlab.ethz.ch/mnowack/slicing-n-cube.

Unfortunately, the link to the source code and intermediate results of the old computer program used to assert $f_{\text{SC}}(5) = 5$ is dead and the authors did not respond when contacted by email. This makes a direct comparison with their implementation impossible.

4.1 Slicing the 5-cube

We implemented the algorithm outlined in Chapter 3 for any n from scratch in C++. The paper, which was published in 2000, reports a running time of two months for $n = 5$. Our program runs in less than 90 minutes for $n = 5$ on an Intel® Xeon® Processor E3-1284L v4 @ 2.90GHz with 16 GB of RAM.

We are encoding sliceable sets as bitstrings of length $|E_n|$ where the i-th bit is 1 iff the i-th edge (according to some fixed enumeration of the edges) is in this sliceable set. The union of two sliceable sets is then equivalent to the bitwise OR of the bitstrings. Similarly, checking if a sliceable set equals E_n is equivalent to testing if the bitstring is all ones.

The encoding of sliceable sets as bitstrings enables a major improvement to Algorithm 3. In its canonical form it sorts the M2SSs with respect to cardinality and then ignores all pairwise unions where $|A| + |B| < |E_n|$ for a USR-M2SS A and a M2SS B because such a union can anyway not be equal to E_n. However, sorting the M2SSs with respect to their lexicographic order establishes a much stronger bound. Recall that a USR is defined as the lexicographically smallest member of its equivalence class. Thus, we expect that the bitstring encoding of a USR-M2SS A has many leading zeros. Since we are looking for two bitstrings whose bitwise OR is all ones, we can ignore all M2SS B which have less leading ones than A has leading zeros. At $n = 5$

n	2	3	4	5	6	7
$\|E_n\|$	4	12	32	80	192	448
$\|\Sigma_n\|$	8	48	384	3840	46 080	645 120
#MSSs	6	51	940	47 285	7 514 066	4 189 035 431
#USR-MSSs	2	5	14	62	566	14 754
#M2SSs	1	92	82 336	$\leq$567 584 264	??	??
#USR-M2SSs	1	5	301	$\leq$167 665	??	??

Table 4.1: Sizes of intermediate results

there are $9.5 \cdot 10^{13}$ pairwise unions. Sorting by cardinality reduces the unions that have to be considered to $6.5 \cdot 10^{13}$ whereas sorting by lexicographic order reduces them to $9.1 \cdot 10^{10}$. This reduces the computational complexity of the algorithm by over a factor of 700 compared to the original algorithm!

4.2 Slicing the 7-cube

We would have like to run the same program for $n = 7$ and have it terminate in reasonable time. Unfortunately, we can argue that extending this algorithm to $n = 7$ is infeasible merely by considering memory requirements. Since sliceable sets are encoded as bitstrings of length $|E_n|$, each sliceable set requires $\lceil |E_n|/8 \rceil$ bytes of memory. For $n = 5$ the ratio between the number of MSSs and M2SSs is $\approx$ 12000. Even under the hopelessly optimistic assumption that for $n = 7$ this ratio remains, 2.8 petabytes will be required to store the M2SSs. A more realistic estimate would likely measure the memory requirements in exabytes. And the USR-M3SSs will be even larger than that. Such an amount of memory is unaffordable at this point in time even if the computational complexity of the program were traceable.

4.3 Sizes of maximal sliceable sets

Table 4.1 displays the sizes of intermediate results reported by our program. The number of MSSs and USR-MSSs matches the numbers reported by Sohler and Ziegler [14]. However, our numbers for the M2SSs and USR-M2SSs for $n = 5$ slightly deviate from theirs (597 175 584 and 175 946). They did not mention the M2SSs or USR-M2SSs of other dimensions. Since their source material is unavailable, we cannot say with certainty where this discrepancy stems from, but it seems plausible that this is simply due to the fact that the set returned by Algorithm 2 contains additional non-maximal 2-sliceable sets. How many superfluous 2-sliceable sets are returned depends on how exactly the USR is defined.

Figure 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 display the number of MSSs broken down by their cardinality for $2 \leq n \leq 7$. The raw data of these figures is also

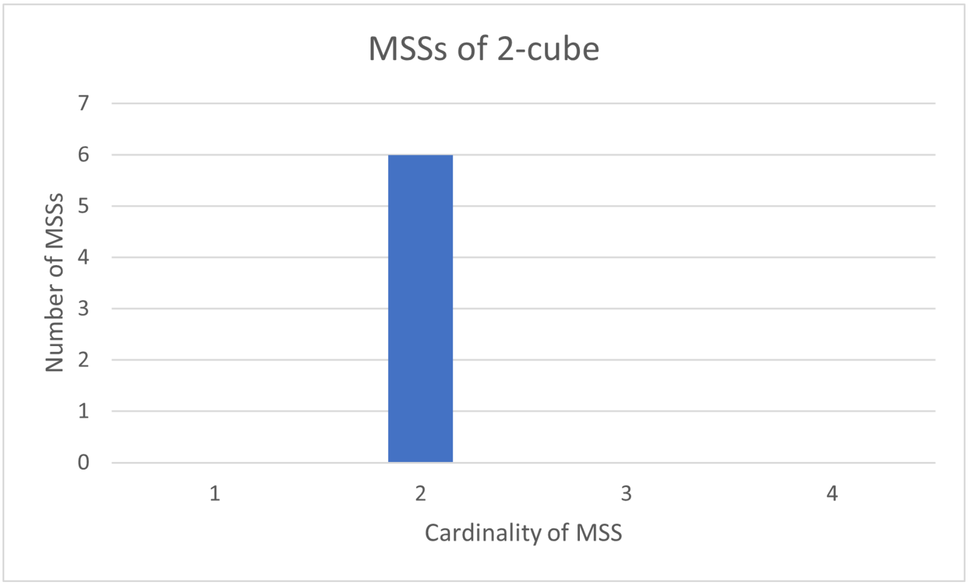

Figure 4.1: Cardinalities of MSSs for $n = 2$

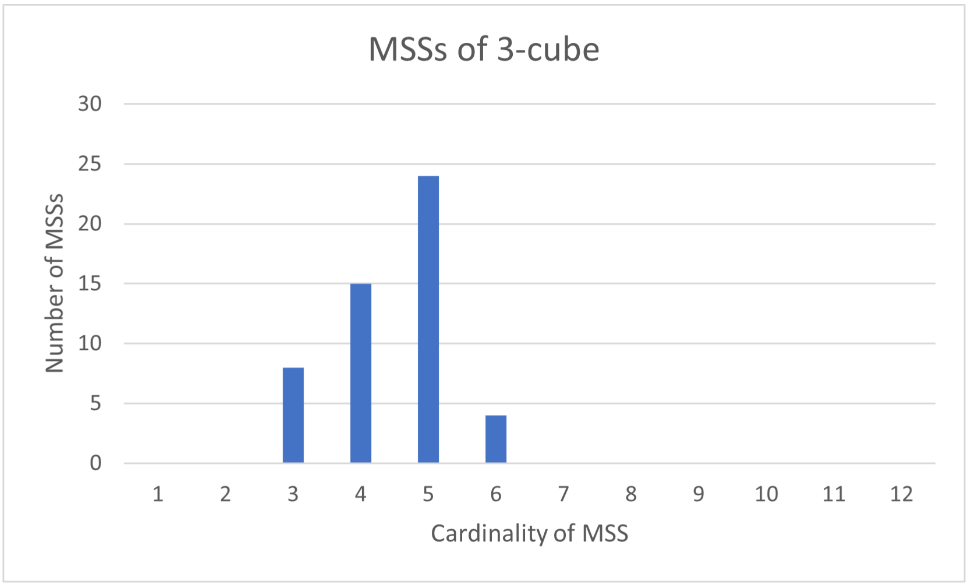

Figure 4.2: Cardinalities of MSSs for $n = 3$

available in our repository. In this small dataset we observe the following.

- Any MSS slices at least n edges.
- The maximal number of edges sliced by a MSS matches Equation 2.1.
- For even n any MSS slices an even number of edges.

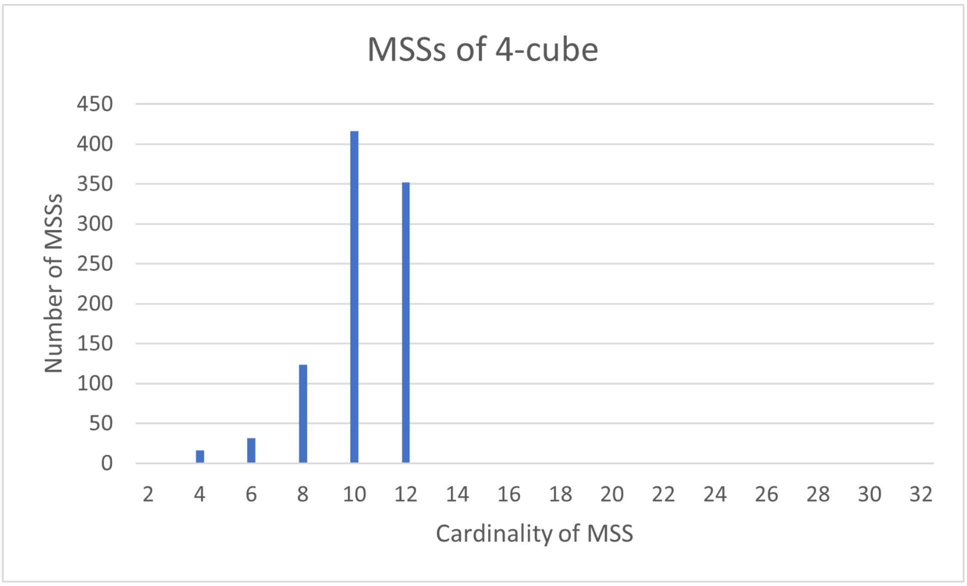

Figure 4.3: Cardinalities of MSSs for $n = 4$

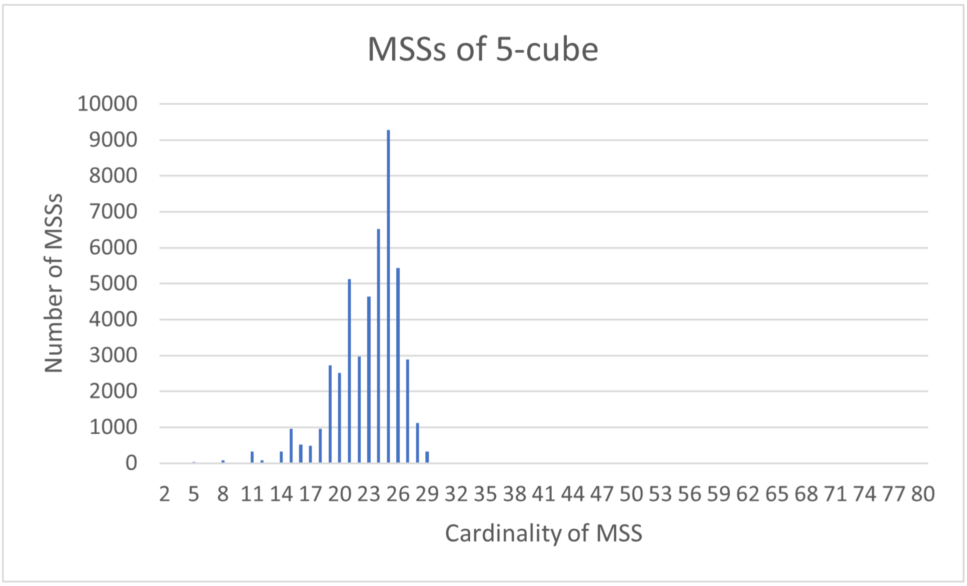

Figure 4.4: Cardinalities of MSSs for $n = 5$

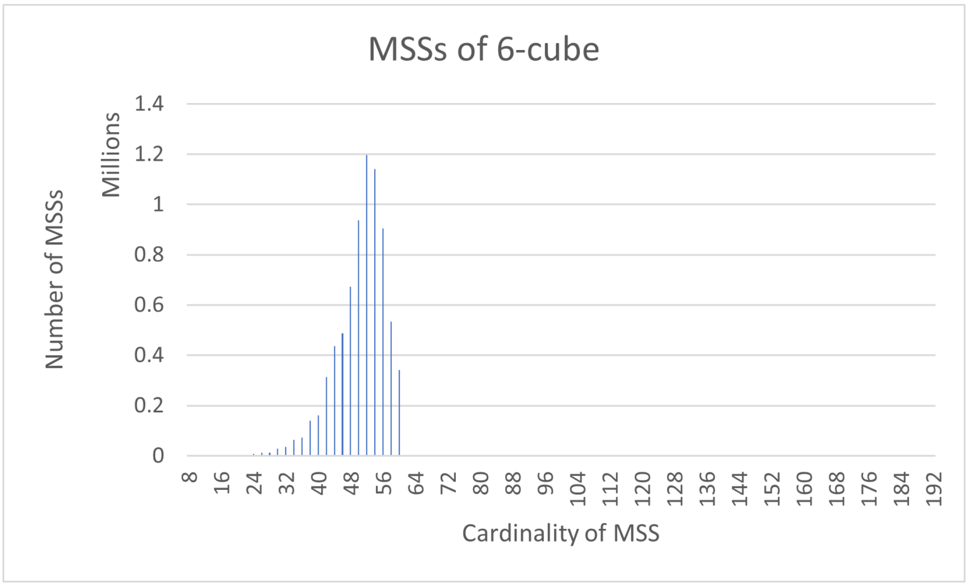

Figure 4.5: Cardinalities MSSs for $n = 6$

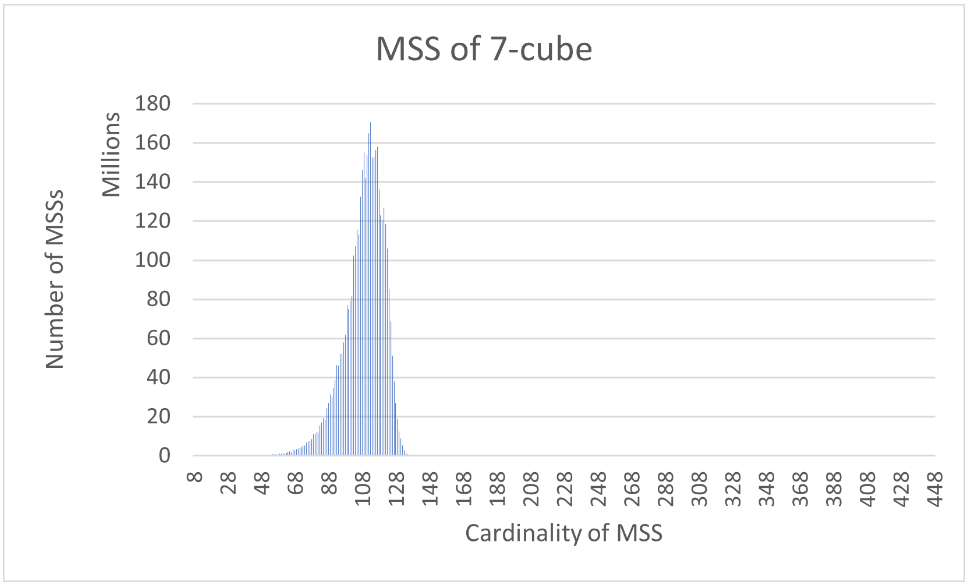

Figure 4.6: Cardinalities of MSSs for $n = 7$

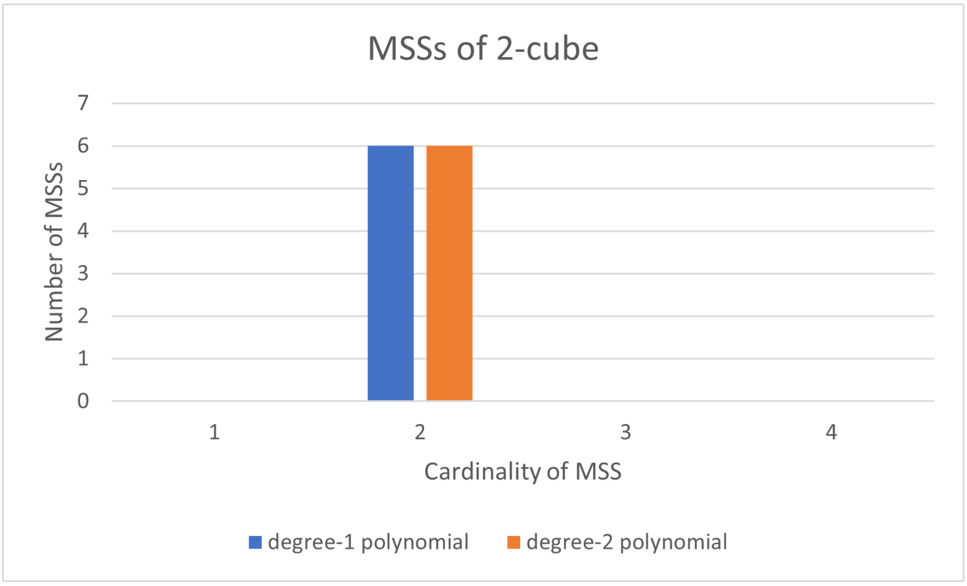

Figure 4.7: Cardinalities of MSSs for $n = 2$

4.4 Degree-2 polynomials

Extending the linear program in Algorithm 1 (Equation 3.1) to higher degree polynomials is straightforward. However, the correctness of the algorithm relies on the assumption that the maximal sliceable sets correspond exactly to the exterior edges of cut complexes. It is not obvious if this still holds for higher degree polynomials. Nevertheless, it is clear that all sliceable sets returned by the Algorithm 1 are still valid sliceable sets. The uncertainty is whether the returned sliceable sets contain all maximal sliceable sets.

Subject to this uncertainty, we find that while at least n degree-1 polynomials are needed to slice all the edges of the n-cube for $n \leq 5$, this bound drops to 2 for $n = 3$, 2 for $n = 4$, and 3 for $n = 5$ when using degree-2 polynomials. For $n = 2$ the bound remains 2. Furthermore, Figure 4.7, 4.8, 4.9, and 4.10 illustrate the difference in the number of MSSs and their cardinality between degree-1 and degree-2 polynomials.

4.5 Low-weight hyperplanes

If all normal vectors have coefficients ± 1, then at least $\frac{n}{2}$ hyperplanes are needed to slice all the edges of the n-cube [1, 12]. It would be interesting to know the exact number of hyperplanes needed to slice all the edges of the n-cube for concrete n in this setting.

The following lemma enables an exhaustive search over all potential hyperplanes whose normal vector has only integer coefficients.

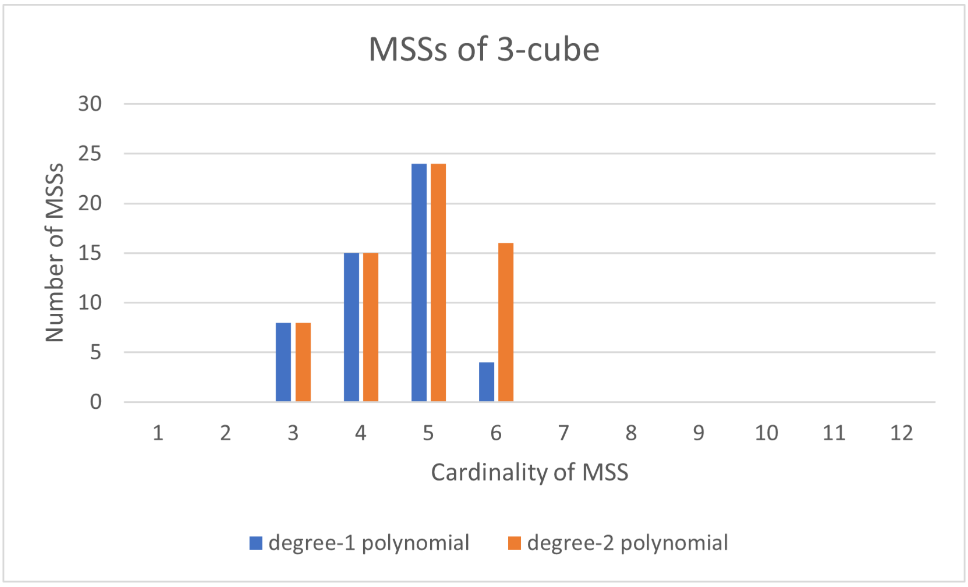

Figure 4.8: Cardinalities of MSSs for $n = 3$

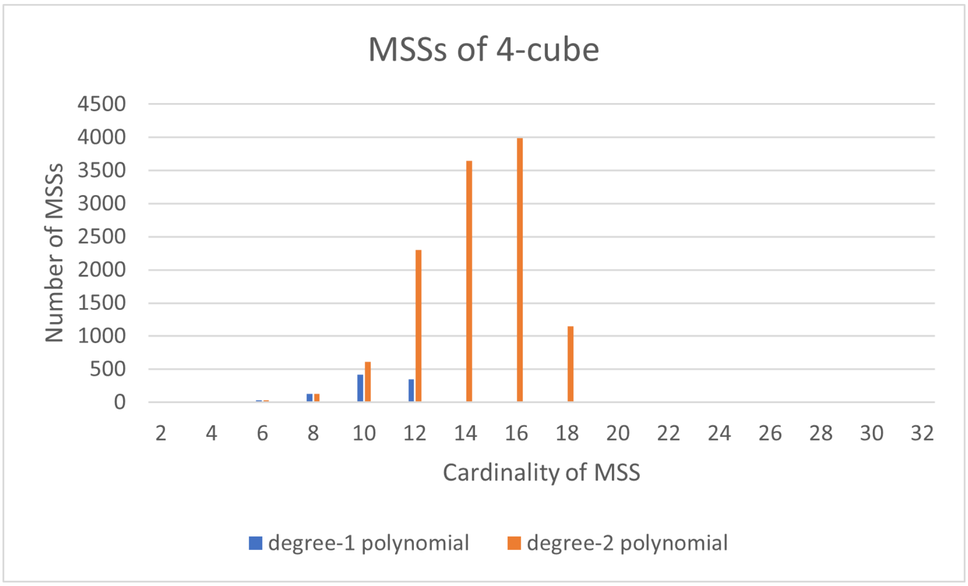

Figure 4.9: Cardinalities of MSSs for $n = 4$

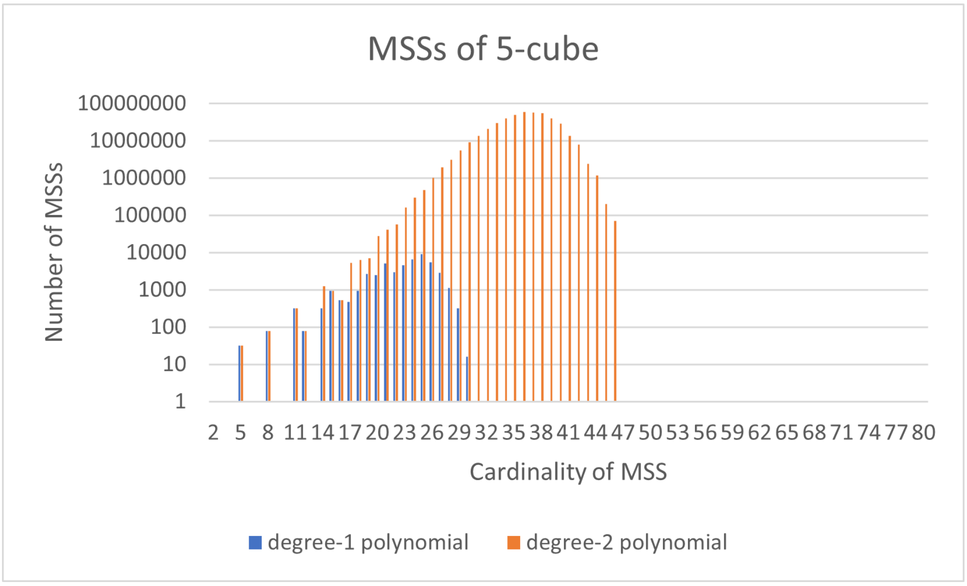

Figure 4.10: Cardinalities of MSSs for $n = 5$ (logarithmic y-axis)

Lemma 4.1 *If the normal vector u of a hyperplane $H = \{w \in \mathbb{R}^n : w \cdot u = t\}$ has only integer coefficients in $\{-m, \ldots, m\}$, it suffices to consider integer thresholds t where $0 \leq t < n \cdot m$ for the purpose of slicing edges of the n-cube.*

Proof Negative thresholds do not have to be considered because

$$H = \{w \in \mathbb{R}^n : w \cdot u = t\} = \{w \in \mathbb{R}^n : w \cdot -u = -t\}.$$

Next, recall that an edge (x, y) of the n-cube is sliced by a hyperplane if

$$x \cdot u < t \wedge y \cdot u > t$$

or

$$x \cdot u > t \wedge y \cdot u < t$$

and that $x, y \in \{-1, 1\}^n$.

Thresholds $t \geq n \cdot m$ do not have to be considered because $x \cdot u \leq nm$ for all vertices x and hence a hyperplane with such a threshold slices no edges.

Finally, since the two vertices that form an edge (x, y) differ in exactly one coordinate, $|x \cdot u - y \cdot u| \geq 2$. Thus, there always exists an integer threshold between $x \cdot u$ and $y \cdot u$ that a real-valued threshold can be rounded towards.$\square$

Since there are only 2^n normal vectors with coefficients ± 1, it is feasible for $n \leq 7$ to generate all corresponding sliceable sets and input them to our program to compute the minimum number of hyperplanes needed to slice all the edges of the n-cube. We find that the minimum in this setting is

n	2	3	4	5	6
#MSSs	6	51	940	47 285	7 514 066
#MSSs$_{m=1}$	6	27	108	405	1 458
#MSSs$_{m=2}$		51	492	4 165	29 938
#MSSs$_{m=3}$			940	16 405	202 386
#MSSs$_{m=4}$				31 285	682 002
#MSSs$_{m=5}$				47 285	??

Table 4.2: Number of maximal sliceable sets in the general setting and when the normal vectors have only integer coefficients in $\{-m, \dots, m\}$
.

always n. For $n \leq 5$ this is actually trivial because we know $f_{SC}(n) = n$, but for $n \in \{6, 7\}$ it is interesting to observe that the answer is not 5 or 6. Interestingly enough, the answer remains n if we additionally restrict the threshold to $\{0, 1\}$ rather than $\{0, \dots, n-1\}$.

When considering hyperplanes whose normal vectors have only integer coefficients in $\{-m, \dots, m\}$, two interesting questions present themselves.

1. What is the smallest m so that the minimum number of these hyperplanes needed to slice all the edges n-cube is equal to $f_{SC}(n)$?
2. What is the smallest m so that the set of all sliceable sets induced by these hyperplanes is equal to the set of all sliceable sets in the general setting?

The answer to the first question is 1 when $n \leq 5$ as already determined by the above results for normal vectors with coefficients ± 1. Determining the answer for $n > 5$ is unfortunately computationally infeasible.

The answer the second question can be determined computationally for small enough n because the total number of normal vectors to consider is $(2m+1)^n$. In Table 4.2 we can that see that the answer is 1 for $n = 2$, 2 for $n = 3$, 3 for $n = 4$, and 5 for $n = 5$. Furthermore, we can safely assume on the basis of the intermediate results that the answer is at least 7 for $n = 6$.

4.6 Further optimizations to the program

Our implementation puts an emphasis on simplicity and makes heavy use of the C++17 standard library and follows a functional programming paradigm. This makes the source code easy to follow and increases confidence in the correctness of the code. However, this also comes at the expense of some performance. On the other hand, this is of limited practical implications due to the inherently exponential problem complexity. Nevertheless, we will discuss in the following some noteworthy constant time improvements applicable to our implementation.

4.6.1 Pure C

In the C++ implementation we use the `std::bitset` data structure offered by the standard library to encode sliceable sets. While this data structure implements most desired functionality, it does not expose the underlying bytes to the user and, for some inexplicable reason, comparators are not defined by the C++ standard. We can of course implement comparators ourselves, but this is undesirable because we have to compare every bit individually instead of comparing 64 bits at once (or even more using vector intrinsics). This issues can be circumvented by storing the sliceable sets as raw bytes in memory and directly calling bitwise operators on the bytes.

Concretely, we re-implemented in pure C the final step of the algorithm for the specific case $n = 5$, i.e. checking if any of the pairwise unions of all M2SSs and USR-M2SSs equals E_5. We chose this part because it used to be the bottleneck prior to the optimization described in Section 4.1. After the discovery of this optimization, the C implementation became of limited use because the new bottleneck shifted to the previous step in the algorithm, i.e. computing the USR-M2SSs. Nevertheless, it nicely illustrates the speed up that can be gained by a pure C implementation. The C++ implementation takes 930 seconds finish this part, while the C implementation finishes in under 90 seconds. This is a speed up of more than a factor of 10! However, we will point out that the new bottleneck of the program, computing the USR of a sliceable set, is less likely to benefit from a pure C implementation because this operation inherently operates on individual bits, rather than contiguous bits.

4.6.2 Caching symmetries

The remaining bottleneck of the program, computing the USR of a sliceable set, generates all possible symmetries and applies them to the vertices of the edges. Instead of generating and applying all symmetries every time a USR is computed, all vertex and/or edge transformations can be precomputed once. This yields a speedup of a factor of 2-4. However, we ultimately abandoned this optimization to avoid passing the precomputed transformations as argument to all the functions and more importantly to avoid the incurred memory overhead (the symmetries can be generated one by one in-place). For $n = 7$ it already takes 1.5 GB to store all vertex and edge transformations. This turns computing the USR of a sliceable set into a memory bound task which conflicts with the much more significant speed up achievable by parallelization.

4.6.3 Parallelization

The remaining bottleneck of the algorithm, computing the USR of a sliceable set, is conceptually well-suited for parallelization. Split the set of all symmetries into m balanced sets, let m threads compute the lexicographically smallest transformation amongst their assigned set and finally compute the lexicographically smallest transformation among the m transformations returned by each thread. However, as noted in the previous section, we observed that when each thread operates on a subset of the precomputed edge transformations, the speedup from parallelization diminishes very quickly, presumably because the running time is dominated by fetching the precomputed edge transformations from shared memory. If the edge transformations are not already precomputed in a convenient data structure, splitting the symmetries into m balanced sets becomes more complicated. One way to approach this in our implementation would be to spawn $m = n^c$ threads where each thread receives a sequence $s \in [n]^c$. Then, each thread generates all symmetries whose permutation has prefix s and computes the lexicographically smallest transformation amongst them.

Bibliography

[1] Noga Alon, Ernest E Bergmann, Don Coppersmith, and Andrew M Odlyzko. Balancing sets of vectors. *IEEE Transactions on Information Theory*, 34(1):128–130, 1988.

[2] Kirby A Baker. A generalization of sperner's lemma. *Journal of Combinatorial Theory*, 6(2):224–225, 1969.

[3] Ilias Diakonikolas, Daniel M Kane, Vasilis Kontonis, Christos Tzamos, and Nikos Zarifis. Agnostic proper learning of halfspaces under gaussian marginals. *arXiv preprint arXiv:2102.05629*, 2021.

[4] Ilias Diakonikolas, Daniel M Kane, Vasilis Kontonis, Christos Tzamos, and Nikos Zarifis. Learning general halfspaces with general massart noise under the gaussian distribution. *arXiv preprint arXiv:2108.08767*, 2021.

[5] M Reza Emamy-Khansary. On the cuts and cut number of the 4-cube. *Journal of Combinatorial Theory, Series A*, 41(2):221–227, 1986.

[6] Johan Håstad. On the size of weights for threshold gates. *SIAM Journal on Discrete Mathematics*, 7(3):484–492, 1994.

[7] Daniel Kane. The average sensitivity of an intersection of half spaces. *Research in the Mathematical Sciences*, 1(1):1–8, 2014.

[8] Adam Klivans and Pravesh Kothari. Embedding hard learning problems into gaussian space. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[9] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[10] Patrick E O'neil. Hyperplane cuts of an n-cube. *Discrete Mathematics*, 1(2):193–195, 1971.

[11] M. Paterson. unpublished; see example 3.72 in [12].

[12] Michael Saks. Slicing the hypercube. *Surveys in combinatorics*, 1993:211–255, 1993.

[13] Alexander A Sherstov and Pei Wu. Near-optimal lower bounds on the threshold degree and sign-rank of ac ˆ0. *SIAM Journal on Computing*, (0):STOC19–1, 2021.

[14] Christian Sohler and Martin Ziegler. Computing cut numbers. 2000.

[15] Roei Tell. Quantified derandomization of linear threshold circuits. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of* $f_{\mathrm{SC}}(n)$*Computing*, pages 855–865, 2018.

[16] Gal Yehuda and Amir Yehudayoff. Slicing the hypercube is not easy, 2021.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Slicing the n-cube

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Nowack	Manuel

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 21.01.2022	M. Nowack

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.