9 Bounds for the Knapsack Problem (March 6)

In this lecture, I'll develop both upper and lower bounds in the linear decision tree model for the following version of the (NP-complete) KNAPSACK[1] problem: Given a set of n real numbers, does any subset add up to 1?

9.1 A Quadratic Lower Bound

To prove a lower bound, we use the component-counting method introduced last week. Let KS_n be the set of points in $\mathbb{R}^n$ such that some subset of the coordinates sums to 1. It's not hard to check that KS_n is connected. The lower bound will follow directly from the following lemma:

Lemma 1. *$\mathbb{R}^n \setminus KS_n$ has at least $2^{\binom{n}{2}+1}$ connected components.*

Proof: For any point $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n \setminus KS_n$, we can associate a boolean function $f_x : \{0,1\}^n \to \{0,1\}$ defined as follows:

$$f_x(a_1, a_2, \ldots, a_n) = \left[\sum_{i=1}^{n} a_i x_i > 1\right]$$

In other words, $f_x(a) = 1$ if and only if the subset of coefficients of x specified by the bit vector a adds up to more than 1, and $f_x(a) = 0$ if that subset adds up to less than 1. Two points x and y are in the same connected component of $\mathbb{R}^n \setminus KS_n$ if and only if the corresponding functions f_x and f_y are identical. Thus, to count the connected components of $\mathbb{R}^n \setminus KS_n$, it suffices to count the boolean functions of the form f_x.

An n-bit boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a *threshold function* if there exist real numbers $z_1, z_2, \ldots, z_n, \theta$ such that

$$f(a_1, a_2, \ldots, a_n) = 0 \iff \sum_{i=1}^{n} a_i z_i < \theta.$$

for all bit vectors $(a_1, a_2, \ldots, a_n)$. In other words, there is a hyperplane that separates $f^{-1}(0)$ and $f^{-1}(1)$ (as points in $\mathbb{R}^n$). Let T_n denote the set of n-bit threshold functions $f : \{0,1\}^n \to \{0,1\}$.

The *weight vector* z and the *threshold* θ of a threshold function f are not uniquely determined. In fact, for any threshold function, there is an open n-dimensional set of possible weight vectors, even if we fix the threshold value. For almost all[2] of these weight vectors, the weighted sum

$$\sum_{i=1}^{n} a_i z_i$$

has 2^n different values, one for each bit vector a. Thus, for any threshold function, we can define a family of $2^n + 1$ *parallel* threshold functions with the same weight vector but different threshold values.

Any function of the form f_x is obviously a threshold function with weight vector x and threshold 1. Moreover, if f is a threshold function with $f(0) = 0$, then the threshold θ of f is non-zero,

[1] This is more accurately a version of the SUBSET SUM problem, but it's called KNAPSACK for historical reasons. Specifically, because that's what Dobkin and Lipton called it in their lower bound paper, and it's what Meyer auf der Heide called it in his upper bound paper. I don't know what they were thinking.

[2] This is a technical term meaning "all but a measure-zero subset".

which implies that $f = f_x$, where $x = z/\theta$. if f is a threshold function, then so if $1 - f$; thus exactly half the threshold functions satisfy $f(0) = 0$.

It follows that the number of connected components of $\mathbb{R}^n \setminus KS_n$ is exactly half the number of elements of T_n. To complete the proof, I need to show that $|T_n| \geq 4 \cdot 2^{\binom{n}{2}}$. I'll prove this by induction, with the base case $|T_1| = 4$; *every* 1-bit boolean function is a threshold function.

We have the following bijection between n-bit boolean functions f and pairs (f_0, f_1) of $(n-1)$-bit boolean functions:

$$f(a_1, a_2, \ldots, a_n) = \begin{cases} f_0(a_1, \ldots, a_{n-1}) & \text{if } a_n = 0 \\ f_1(a_1, \ldots, a_{n-1}) & \text{if } a_n = 1 \end{cases}$$

It is not hard to prove that f is a threshold function if and only if f_0 and f_1 are *parallel* threshold functions. Thus, we can specify a unique n-bit threshold function f by choosing an $(n-1)$-bit threshold function f_0 and then choosing another function f_1 parallel to f_0. Since there are $2^{n-1}+1$ threshold functions parallel to any f_0 (including f_0 itself), we have the following recurrence:

$$|T_n| \geq |T_{n-1}| \cdot (2^{n-1} + 1).$$

The inductive hypothesis implies that

$$|T_{n-1}| \geq 4 \cdot 2^{\binom{n-1}{2}},$$

so

$$|T_n| \geq 4 \cdot 2^{\binom{n-1}{2}} \cdot (2^{n-1} + 1) > 4 \cdot 2^{\binom{n-1}{2}+(n-1)} = 4 \cdot 2^{\binom{n}{2}},$$

as claimed. □

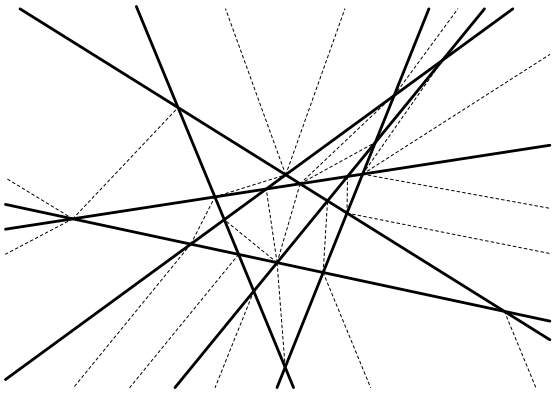
Theorem 2 (Dobkin and Lipton). *Any linear decision tree that solves the* KNAPSACK *problem has depth* $\Omega(n^2)$.

9.2 A Polynomial Upper Bound

To prove an upper bound for KNAPSACK, we will actually use a more general algorithmic result due to Ken Clarkson and Stefan Meiser. Recall that any set of hyperplanes decomposes $\mathbb{R}^d$ into a cell complex called an *arrangement*. For any fixed set of hyperplanes, the *point location* problem asks, given a point in $\mathbb{R}^d$, which cell in the arrangement contains it. The set KS_n is the union of a collection of 2^n hyperplanes in $\mathbb{R}^n$, and the KNAPSACK problem asks whether an input point lies on any of these hyperplanes. Thus, if we can solve the point-location problem quickly, this will give us a fast algorithm for KNAPSACK.

To locate points quickly, it is helpful to consider a refinement of the hyperplane arrangement into simplices. We define the *bottom-vertex triangulation* $\nabla(\Pi)$ of any convex polyhedron Π recursively as follows. If the polyhedron is a line segment, it is already triangulated. Otherwise, let v be the lowest vertex of Π in some fixed direction (breaking ties in some consistent manner).[3] We recursively construct the bottom-vertex triangulations of all the facets of Π, and then decompose Π by joining the lowest vertex v in some fixed direction to the simplices on the facets of Π not containing v. The bottom-vertex triangulation $\nabla(H)$ of the whole arrangements is obtained by combining the bottom vertex triangulations of each of its cells.

[3]We have to define the triangulation carefully when the polyhedron Π is unbounded, since the lowest vertex and/or some of the facets may lie at infinity, but the actual details are not particularly important.



The bottom-vertex triangulation $\nabla(L)$ of an arrangement of lines L.

The following lemma is straightforward (but I won't include the proof).

Lemma 3. *The bottom-vertex triangulation of an arrangement of N hyperplanes in $\mathbb{R}^d$ contains $O(N^d)$ simplices.*

Let's start by describing a brute-force solution to the point location problem.

Lemma 4. *Let H be a fixed collection of N hyperplanes in $\mathbb{R}^d$. Given a point $x \in \mathbb{R}^d$, we can determine which simplex in $\nabla(H)$ contains x in $O(d^2 n)$ time. Moreover, the algorithm can be described by a linear decision tree with depth $O(dn)$.*

Proof: In a single linear test (which requires $O(d)$ time), we can determine whether x lies above, on, or below, a single hyperplane in H. Thus, using n linear tests (in $O(dn)$ time), we can determine the cell Π in the arrangement of H that contains x. To complete the algorithm, we need to locate the simplex in the bottom-vertex triangulation of Π that contains x.

Let v be the bottom vertex of Π, and let Δ be the simplex in $\nabla(\Pi)$ that contains x. Imagine shooting a ray from v through x to a facet of Π. We can determine the hyperplane $h \in H$ that this ray hits first using n linear tests (in $O(dn)$ time), breaking ties arbitrarily. Let x' be the intersection of the ray $\overrightarrow{vx}$ with h, and let Π' be the facet of Π contained in h. Finally, let Δ' be the simplex in $\nabla(\Pi')$ that contains x'. By the definition of the bottom-vertex triangulation, Δ' is the facet of Δ opposite v. We can locate Δ' recursively using at most $(d-1)n$ linear tests (in $O(d(d-1)n)$ time).

Altogether, our algorithm uses $(d+1)n$ linear tests, and can be executed in $O(d^2 n)$ time. □

To get a faster algorithm, we will use a randomized divide-and-conquer strategy. The actual point-location algorithm is not randomized; all of the randomness lies in the preprocessing stage. In effect, we will construct a large collection of point-location algorithms and argue that at least half of them have small running time. To keep the analysis simple, I'll assume that the hyperplanes H are in *general position*: at most d hyperplanes pass through any point. Although the hyperplanes in the KNAPSACK problem are not in general position, the necessary modifications are simple (but tedious).

The preprocessing begins by choosing a random subset $R \subset H$ of size r, where each r-subset is chosen with equal probability. We then construct the bottom vertex triangulation $\nabla(R)$ of the arrangement of R. For each simplex $\triangle \in \nabla(R)$, we define two sets of hyperplanes:

- $T(\triangle)$ is the set of hyperplanes in H that contain the vertices of $\triangle$. These are called the *triggers* of $\triangle$.

- $S(\triangle)$ is the set of hyperplanes in H that intersect the interior of $\triangle$. These are called the *stoppers* of $\triangle$.

The point-location algorithm determines the simplex $\triangle \in \nabla(R)$ containing the query point x, using the brute-force algorithm described earlier, and then recursively locates x in the bottom-vertex triangulation of $T(\triangle) \cup S(\triangle)$. The claim is that with non-zero probability, this subset of hyperplanes will be a constant factor smaller than H, so after $O(\log n)$ levels of recursion, the algorithm will terminate.

Let $t = t(d)$ denote the maximum number of triggers for a simplex $\triangle \in \nabla(R)$. We need d hyperplanes to determine the bottom vertex, one hyperplane to contain the top facet, and then $t(d-1)$ additional hyperplanes through vertices of the top facet. Thus, we have the recurrence $t(d) = d + 1 + t(d-1)$. With the base case $t(1) = 2$, we have $t = t(d) = d(d+3)/2$. Not all simplices require this many triggers; for example, if a cell in the arrangment is already a simplex, that simplex has only d triggers.

The same simplex $\triangle$ might show up in the bottom-vertex triangulations of several different random samples. Specifically, a simplex $\triangle$ belongs to the bottom-vertex triangulation $\nabla(R)$ if and only if every hyperplane in $T(\triangle)$ is in R, but no hyperplane in $S(\triangle)$ is in R. Also, any set T of t hyperplanes contains the trigger set for all the simplices in $\nabla(T)$; Lemma 3 implies that there are at most $O(t^d)$ such simplices.

We say that a simplex $\triangle$ is *dangerous* if $|S(\triangle)| > \frac{t \ln r}{r} n$, and that our random sample R is *safe* if $\nabla(R)$ contains no dangerous simplices.

We can overestimate the probability that R is unsafe as follows:

$$\begin{aligned}
\Pr[R \text{ is unsafe}] &\leq \sum_{\text{dangerous } \triangle} \Pr[\triangle \in \nabla(R)] \\
&= \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R \ \wedge \ S(\triangle) \cap R = \varnothing] \\
&= \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] \cdot \Pr[S(\triangle) \cap R = \varnothing \mid T(\triangle) \subseteq R]
\end{aligned}$$

We can bound the conditional probability on the right by imagining the following experiment. Having chosen the t triggers of some dangerous simplex $\triangle$, we consider the probability that none of the remaining $r - t$ random draw from H selects a stopper.

$$\begin{aligned}
&= \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] \cdot \prod_{i=t(d)+1}^{r} \left(1 - \frac{|S(\triangle)|}{n-i}\right) \\
&< \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] \cdot \left(1 - \frac{|S(\triangle)|}{n}\right)^{r-t} \\
&< \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] \cdot e^{-|S(\triangle)|r/n} \\
&< \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] \cdot e^{-t \ln r} \\
&= \sum_{\text{dangerous } \triangle} \Pr[T(\triangle) \subseteq R] / r^t
\end{aligned}$$

Now we can overestimate this sum by considering *all* possible simplices instead of just dangerous simplices.

$$\Pr[R \text{ is unsafe}] \leq \sum_{\triangle} \Pr[T(\triangle) \subseteq R]/r^t$$

Except for the r^t factor, this sum describes the expected number of simplices that are triggered by the random sample R. Since every subset of R of size t triggers $O(t^d)$ simplices, we have

$$\Pr[R \text{ is unsafe}] \leq \binom{r}{t} \frac{O(t^d)}{r^t} < \frac{r^t}{t!} \cdot \frac{O(t^d)}{r^t} = \frac{O(t^d)}{t!} \ll 1.$$

Thus, a *random* sample R is safe with non-zero probability. This immediately implies that at least one r-element subset R is safe.

In the actual algorithm, we choose $r \approx 2t \ln t = O(d^2 \log d)$, so that in any safe sample, every subproblem contains at most $n/2$ hyperplanes.

Theorem 5 (Meiser). *Let H be a fixed collection of N hyperplanes in $\mathbb{R}^d$. Given a point $x \in \mathbb{R}^d$, we can determine which simplex in $\nabla(H)$ contains x in $O(d^4 \log d \log n)$ time.*

Proof: The algorithm uses a safe subset R of the hyperplanes of size $O(d^2 \log d)$. We determine the simplex $\triangle \in \nabla(R)$ that contains the query point x in time $O(d^4 \log d)$ using the brute-force algorithm. Then we recursively locate x in $\nabla(T(\triangle) \cup S(\triangle))$. Because R is safe, this subset of hyperplanes has size at most $n/2$. Thus, after $O(\log n)$ levels of recursion, we are left with only a constant number of hyperplanes, which we can check by brute force. □

For the KNAPSACK problem, we have 2^n hyperplanes in $\mathbb{R}^n$.

Corollary 6 (Meyer auf der Heide). *For any fixed n, the n-dimensional* KNAPSACK *problem can be solved in $O(n^5 \log n)$ time.*