CSE533: Information Theory in Computer Science	September 28, 2010
Lecture 6	
Lecturer: Anup Rao	*Scribe: Lukas Svec*

1 A lower bound for perfect hash functions

Today we shall use graph entropy to improve the obvious lower bound on good hash functions.

Definition 1 (k-perfect hash functions)**.** *Given a family of functions $\mathcal{H} = \{h : [N] \to [b]\}$, we say that $\mathcal{H}$ is a* k-perfect hash family*, if $\forall\ S \subset [N]$, $|S| = k$, there exists $h \in \mathcal{H}$ such that h is injective on S.*

Such families are very useful in computer science. Imagine that $[N]$ is the set of all possible videos of a certain size (a huge set), and think of k as being a much smaller number: the total number of videos we want to store in a database. Now suppose $k-1$ videos have already been stored. One way to check whether a new video is in the database or not is to compare the bits of the new video with every stored video, which would be computationally expensive. If one has access to a k-perfect hash family, we can instead have a different server maintaining a hash table for each of the hash functions in the family, and each such server can check whether the new video matches up with a stored video or not. The k-perfect property guarantees that some server in the family will identify the video as being new. In addition, all of the computations can be done in parallel.

Let $t = |\mathcal{H}|$ be the size of the k-perfect family. How small can t be?

Claim 2. $t \geq \log N / \log b$.

Proof This is an application of the pigeonhole principle. Suppose $t < \log N / \log b$. Then $b^t < N$, so two elements of $[N]$ must be hashed in exactly the same way by every hash function. Thus, the family is not even 2-perfect in this case. ■

Claim 3. *If $b \geq 100k^2$, then there is a k-perfect hash function family of size $t = \mathcal{O}(k \log N)$.*

Proof Idea Pick t random functions and let them be in the family. Then for any fixed set of S of k-elements, each function is injective on the with constant probability. By the Chernoff bound, the probability that no function in the family is injective is at most $\exp(-\Omega(t))$. The total number of such sets S is at most N^k, so by the union bound, the probability that any set does not get an injective function is at most $N^k \exp(-\Omega(t))$, which can be made to be less than t for t as small as in the claim. ■

The following theorem improves the bound given by the pigeonhole principle:

Theorem 4. $t \geq \left(\frac{b^{k-1}}{b(b-1)\cdots(b-k+2)}\right) \frac{\log(N-k+2)}{\log(b-k+2)}$.

The theorem was first proved my Fredman and Komlós [1]. The graph entropy based interpretation of the proof that we discuss here is due to Körner [3]. Today we shall prove the theorem in the case that $b|N$, though essentially the same proof works in the general setting.

Let G denote the following graph:

$$\text{Vertices of } G\text{: } \{(D,x) : D \subset [N] \text{ with } |D| = k-2, x \in D - [N]\}$$
$$\text{Edges of } G\text{: } \{\{(D,x_1),(D,x_2)\} : x_1 \neq x_2\}$$

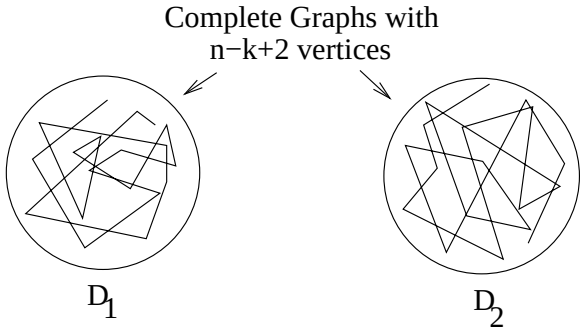
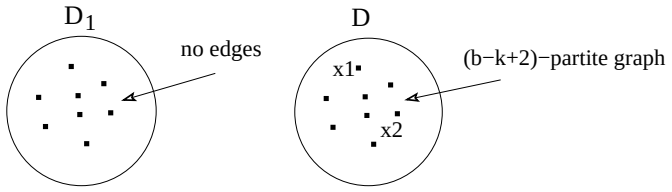


Figure 1: The graph G.

G consists of several connected components, one for every value of D, and each of these connected components is a complete graph on $N-k+2$ vertices. Thus, the graph entropy of each component is $\log(N-k+2)$, and the graph entropy of their union is just a convex combination of the graph entropy of each part, so it is also $H(G) = \log(N-k+2)$.



(a) h is not injective on D_1

(b) h is not injective on D_1

Figure 2: The structure of G_h.

Fix a k-perfect hash function family $\mathcal{H}$. For every $h \in \mathcal{H}$, define a subgraph $G_h \subset G$ with the following subset of edges:

$$\text{Edges of } G_h\text{: } \{\{(D,x_1),(D,x_2)\} : h \text{ is injective on } D \cup \{x_1\} \cup \{x_2\}\}.$$

If D is such that h is *not* injective on D, then G_h has no edges in the component corresponding to D. On the other hand, if h is injective on D, we can partition the vertices (D,x) of the component corresponding to D into $b-k+3$ sets depending on the value of $h(x)$. For every $i \notin h(D)$, define the set $A_i = \{(D,x) : h(x) = i\}$. The component corresponding to D is the $(b-k+2)$-partite graph

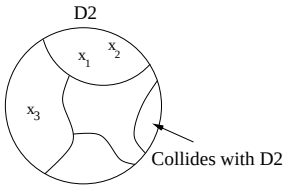


Figure 3: The regions in D_2 are partitioned by h^{-1}.

obtained by partitioning the vertices into the sets A_i. The graph entropy of each such component is thus bounded by $\log(b-k+2)$.

Further, the fact that $\mathcal{H}$ is perfect translates to

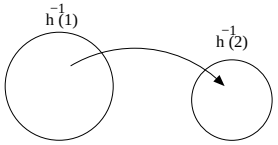
$$G = \bigcup_{h \in \mathcal{H}} G_h.$$

This already gives us a lower bound, since $H(G) \leq \sum_{h \in \mathcal{H}} H(G_h)$, it must be the case that $H(G) \leq t \log(b-k+2)$, in other words, $t \geq \log(N-k+2)/\log(b-k+2)$.

To get a better lower bound, we shall give a better upper bound on $H(G_h)$. The fact that we haven't yet exploited is that many of the vertices of G_h are isolated vertices. A vertex (D, x) is isolated in G_h exactly when h is not injective on $D \cup \{x\}$. By the lemma about the graph entropy of a union of connected components, we know that $H(G_h) \leq \Pr[h \text{ is injective on } D \cup \{x\}] \log(b-k+2)$, where the probability is taken over a random vertex (D, x). This probability is exactly equal to $\Pr[h \text{ is injective on } S]$, for a random set S of size $k-1$. Recall that we are assuming here that $b|N$.

Claim 5. $\Pr[h$ *is injective on* $S]$ *is maximized when* $|h^{-1}(1)| = |h^{-1}(2)| = ... = |h^{-1}(b)|$.

Sketch of Proof



Suppose $h^{-1}(1) > h^{-1}(2) + 1$. Let x be an element with $h(x) = 1$, and let us see what happens to $\Pr_S[h \text{ is injective on } S]$ when we change the value of $h(x)$ from 1 to 2. We can write

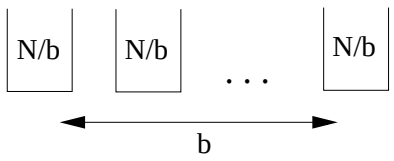
$$\Pr[h \text{ is injective on } S] = \Pr[x \in S] \cdot \Pr[h \text{ is injective on } S | x \in S] + \Pr[x \notin S] \cdot \Pr[h \text{ is injective on } S | x \notin S]$$

Observe that the second term in the sum is unaffected by changing the value of $h(x)$, and the first term can only increase. Thus, $\Pr[h \text{ is injective on } S]$ is maximized when all preimages are of the same size.

■

Next we bound the probability in the case that all preimages are the same size.

Claim 6. $\Pr[h$ *is injective on* $S] \leq \frac{b(b-1)...(b-k+2)}{b^{k-1}}$.



Proof

By Claim 5, it suffices to prove the bound when all preimages of h are of size N/b.

We can think of the elements of $[N]$ as partitioned into b buckets. We need to bound the probability that a random subset of $k-1$ elements gets split into $k-1$ different buckets. The total number of ways of picking $k-1$ elements is $\binom{N}{k-1}$. The number of ways of picking elements from separate buckets is $\binom{b}{k-1}(N/b)^{k-1}$. Thus,

$$\begin{aligned}\Pr[h \text{ is injective on } S] &\leq \frac{\binom{b}{k-1}(N/b)^{k-1}}{\binom{N}{k-1}} \\ &= \frac{b(b-1)\cdots(b-k+2)}{b^{k-1}} \cdot \frac{N^{k-1}}{N(N-1)\cdots(N-k+2)} \\ &\leq \frac{b(b-1)\cdots(b-k+2)}{b^{k-1}}\end{aligned}$$

Thus, $H(G_h) \leq \frac{b(b-1)\cdots(b-k+2)}{b^{k-1}} \log(b-k+2)$, proving the theorem.

■

2 A sorting algorithm

Suppose we are given n integers $x_1, \ldots, x_n$. Then, by the pigeonhole principle we need to make at least $\log(n!) = \Theta(n \log n)$ comparisons to sort them and rearrange them into ascending order, since if we use fewer comparisons, two different permutations of the integers can give the same outcome for every comparison, so we would not be able to distinguish the two. Indeed, we know of several ways to sort using $\Theta(n \log n)$ queries.

Recall that a partial order is a relation $\leq$ on the elements such that $x \leq x$ for all elements, and $x \leq y \leq z$ implies that $x \leq z$. A total order is a partial order in which any two elements are comparable. Every partial order can be completed to a total order.

Question: Given a partial order S on $x_1, ..., x_n$, how many queries are necessary to complete the partial order to a total order?

Define $e(S)$ to be the number of total orders consistent with S. Just as before, it is immediate from the pigeonhole principle that at least $\log(e(S))$ queries are necessary.

Kahn and Kim [2] gave a sorting algorithm matching this lower bound.

Theorem 7. *There a polynomial time algorithm that given S and $x_1, \ldots, x_n$, makes $\log(e(S))$ queries to determine the total order of $x_1, \ldots, x_n$.*

Their algorithm uses graph entropy. Note that unlike the other examples we have seen, they use graph entropy to get an algorithm, rather than a lower bound. Here we sketch the high level outline of their algorithm.

Let G_S denote the comparability graph of S, namely the graph whose vertices are $x_1, \ldots, x_n$, and edges are pairs of elements that are comparable in S. Thus, if S is a total order, G_S is the complete graph, and $H(G_S) = \log n$. On the other hand, if G_S is the empty graph, then S is the empty order.

Kahn and Kim proceed by proving two theorems about the graph G_S. The first gives us an estimate of distance between our graph G_S and the complete graph

Theorem 8. *There is a constant C such that*

$$n(\log n - H(G_s)) \geq \log(e(S)) \geq Cn(\log n - H(G_s)$$

The second theorem states that it is always possible to increase the entropy of the graph G_S by some factor c/n if one picks the right vertices x, y on the graph. Let $S(x \leq y)$ denote the partial order obtained by extending S with the relation $x \leq y$ (and all its consequences).

Theorem 9. *There is a constant c such that if S is not a total order, $\exists\ x, y$ s.t.*

$$\min\{H(G_{S(x\leq y)}, H(G_{S(y\leq x)}\} \geq H(G_S) + \frac{c}{n}$$

$H(G)$ can be computed using a convex program, and there is an polynomial time algorithm that can approximate its value up to a polynomially small error. The algorithm for sorting operates as follows. In each step, the algorithm computes the left hand side of Theorem 9 for every pair x, y, using the current partial order S, and compares the pair that maximizes this value. The query must increase the graph entropy by c/n. After $\log(e(S))/c$ steps, the algorithm must reach a total order by Theorem 8

3 Locally Decodable Codes

Here we sketch a beautiful lower bound for 2-query locally decodable codes, discovered by Alex Samorodnitsky. Rather than give a formal definition, let us start with an example, the famous Hadamard code.

Suppose we want to store an n bit string $x \in \mathbb{F}_2^n$, but we are afraid that some of the bits we store may get corrupted. In the Hadamard code, we would store the value of $\langle a, x\rangle = \sum_{i=1}^n a_i \cdot x_i$ for every $a \in \mathbb{F}_2^n$. This, takes a lot of space — we are storing 2^n bits instead of just n bits. The advantage of this scheme is that it can tolerate many errors, and has a fast decoding algorithm.

3.1 Decoding x_i

To decode a particular bit x_i,

1. Pick $a \in \mathbb{F}_2^n$ uniformly at random.
2. Query $q_1 = \langle a, x\rangle$ and $q_2 = \langle a + e^i, x\rangle$, where $e^i = (0, ..., 0, 1, 0, ..., 0)$, the unit vector in the i'th direction.

3. Output $q_1 + q_2$.

Claim 10. *If at most* 1% *of the stored information is corrupted, we recover* x_i *with probability at least* 98%.

Proof Since both a and $a + e_i$ are uniformly random vectors, the probability that either query is corrupted is at most 2%. If neither is corrupted $\langle a, x\rangle + \langle a + e^i, x\rangle = \langle a + a + e^i, x\rangle = x_i$. ■

This code is a 2-query code, since we only make two queries to recover x_i. The code is linear, since the encoding is a linear function of the input bits.

Theorem 11 (Samorodnitsky)**.** *Every linear* 2*-query locally decodable code must store* $2^{\Omega(n)}$ *bits.*

Let us sketch the proof. Suppose the code encodes x into the vector $\langle a^1, x\rangle, \langle a^2, x\rangle, \ldots, \langle a^m, x\rangle$. It is no loss of generality to assume that the decoder succeeds in decoding x_i only when it queries the code at a^j, a^k such that $a^j + a^k = e^i$. Indeed, if x is uniformly random and the span of a^j, a^k do not contain e^i, then the value of x_i is independent of the queried values, so the queries are useless.

If the code is to tolerate a 1% fraction of errors, then it must be the case that there are at least $m/200$ pairs of bits that can be queried to recover x_i. If not, the value of x_i can be destroyed by randomly corrupting at most $m/100$ of the stored bits.

Recall that the hypercube is the graph whose vertices are $\mathbb{F}_2^n$, and two vertices are adjacent if and only if they disagree in exactly one coordinate. We have just argued that if any linear 2-query code uses only m bits of storage, then there must be a set $S = \{a^1, \ldots, a^m\}$ of vertices in the hypercube, such that for every i, S contains at least $m/200$ edges in direction i. The lower bound will be proved by showing the following claim.

Claim 12. *For every subset* S *of vertices in the hypercube,*

$$\#\text{ of edges in } S \leq \frac{|S|\log(|S|)}{2}$$

The bound is tight when S is a subcube.

Given the claim, the fact that S has $m/200$ edges in each direction implies that

$$\frac{nm}{200} \leq \frac{m\log m}{2} \Rightarrow m \geq 2^{n/100}.$$

The proof of the claim goes by a slick entropy argument.

Proof Let $X_1, ..., X_n$ be a uniformly random vertex in S.

Then observe that for every $x_2, \ldots, x_n$ in the support of $X_1, \ldots, X_n$, $X_1|x_2, \ldots, x_n$ is a uniform bit if there is an edge in direction 1 at $x_2, \ldots, x_n$, and is constant if not.

Thus,

$$H(X_1|x_2, \ldots, x_n) = \begin{cases} 1 & \text{if there is an edge in direction 1 at } x_2, \ldots, x_n, \text{ and} \\ 0 & \text{else.} \end{cases}$$

Therefore, the number of edges in direction i is exactly $|S|H(X_i|X_{-i})/2$, where X_{-i} denotes all coordinate except for the i'th one. The factor of 2 comes from the fact that every edge gets a probability of $2/|S|$ in the expectation.

This allows us to bound the total number of edges by

$$\begin{aligned}
&(|S|/2)\sum_{i=1}^{n} H(X_i|X_{-i}) \\
&\leq (|S|/2)\sum_{i=1}^{n} H(X_i|X_1,\ldots,X_{i-1}) \\
&= (|S|/2)H(X_1,\ldots,X_n) \\
&= \frac{|S|\log|S|}{2}
\end{aligned}$$

■

References

[1] Michael L. Fredman and János Komlós. On the size of separating systems and families of perfect hash functions. *SIAM Journal on Algebraic and Discrete Methods*, 5(1):61–68, March 1984.

[2] Jeff Kahn and Jeong Han Kim. Entropy and sorting. *Journal of Computer and System Sciences*, 51(3):390–399, December 1995.

[3] J. Körner. Fredman-Komlós bounds and information theory. *SIJADM: SIAM Journal on Algebraic and Discrete Methods*, 7, 1986.