MIT 18.996: Topic in TCS: Internet Research Problems **Spring 2002**

Lecture 5 — March 6

Lecturer: Madhu Sudan *Scribe: Nicole Immorlica and Vahab S. Mirrokni*

Guessing Secrets

Listen, Do you want to know a secret,
Do you promise not to tell, whoa oh, oh.
Closer, Let me whisper in your ear,
Say the words you long to hear...

— Beatles,
Abbey Road Studios, London
February 11, 1963

5.1 Motivation

Today we will discuss issues that arise in DNS. First let's review the DNS mechanism.

Figure 5.1. the DNS mechanism

Akamai has thousands of webservers which cash the foo.com web page, and the goal of Akamai's authorative nameserver is to distribute the foo.com load somewhat evenly among these webservers. As nameservers cache IP addresses, each nameserver will direct all its clients to the same IP address. Thus each nameserver drags a load proportional to the number of its clients. However, we do not know the number of clients of a given nameserver. Furthermore, it is not enough to simply count the number of clients because some clients

have more load than others. What we would really like is to find a mapping of nameservers to clients. And to complicate matters, we must remember that a client may use more than one nameserver.

Let's review the information we are given. Akamai's authorative nameserver knows the IP addresses of all the nameservers that submit foo.com queries. The Akamai webservers know the IP addresses of all the clients that request the foo.com webpage. Now how can we discover which clients were directed to each webserver by which nameserver?

1. IP addresses in the same block

   Perhaps we can assume that clients use nameservers whose IP addresses are in the same block as their own. As IP addresses in the same block tend to be geographically close, this assumption seems reasonable. However, it is unfortunately just not true.

2. Direct each nameserver to a unique webserver and see which clients show up

   Clearly this approach works, but there are a myriad of implementation problems. The first major problem is that there are many more nameservers than webservers. Of course, we can try to sidestep this obstacle by repeating this procedure several times, keeping track of the set of clients at each webserver, taking some intersections, etc. Another major problem is that clients aren't continously requesting foo.com content, so in each iteration of this procedure, Akamai must wait a while to see all the clients that show up.

3. Dynamically create a fake link that encodes client IP on the foo.com webpage

   In this solution, the Akamai webserver dynamically adds a one pixel gif to the foo.com webpage with the web address ¡client ip¿.foo.com. As the client renders the webpage, it will make another request to its nameserver for the IP address of ¡client ip¿.foo.com. When its nameserver asks Akamai's authorative nameserver to resolve this address, the authorative nameserver can associate this client to this nameserver.

   This solution solves our problem and appears to work even though it creates a bit of additional traffic. However, each nameserver now needs to make a unique query for every of its clients. Even if the time-to-live of the reply is zero, some faulty nameserver implementations will try to cache all these unique queries. This floods the cache of the nameserver, and, in some cases, actually crashes the nameserver. While this may appear to be the nameserver's problem, Akamai needs to market its solutions and it is hard to market a solution that breaks existing albeit faulty infrastructure.

4. Statically force a client to reveal each bit of his nameserver's IP address

   In this solution, Akamai adds 32 one pixel gifs to the foo.com webpage with the addresses i.foo.com for $0 \leq i < 32$. Akamai also designates 64 webservers, i_j for $0 \leq i < 32$, $j \in \{0, 1\}$, to host the one pixel gifs. The Akamai authorative nameserver answers a query for i.foo.com with webserver i_j where j is the ith bit of the IP address of the questioning nameserver. By observing which of the 64 webservers a client visited, we can now reconstruct the client's nameserver's IP.

This solution solves the problems of the last solution because now each nameserver will cache just a constant number (i.e. 32) of IP addresses, namely i.foo.com for $0 \leq i < 32$. However, this solution no longer solves our problem when a client uses more than one nameserver.

5.2 Problem Statement

Notice in the last solution, what we really do is ask the client 32 yes/no questions regarding his nameservers' IP addresses. Namely, we ask 32 questions of the form "What is the ith bit of your nameserver's IP address?". The client then selects one of his nameservers and answers our question truthfully. This strategy is a little bit naive and also quite weak. For example, if the client had a nameserver with an all-zeros IP address and a nameserver with an all-ones IP address, he could answer our 32 questions in any of the 2^{32} possible ways. Thus this strategy allows us to reach no conclusion regarding his nameservers' IP addresses. In this section, we will extend this strategy by asking more questions. In this way, we hope we will be able to conclude more information about the possible nameservers of the client.

First we give a formal definition of our problem:

Definition 1. The Guessing Secrets Problem: *Alice (the adversary) has a set $S = \{X_1, \ldots, X_k\}$ of k secrects, taken from a universe Ω of size N. Bob asks Alice a series of "questions" $\{F_i | F_i : \Omega \to \{0,1\}\}$. For each F_i, Alice selects some $X_i \in \Omega$ and returns $\{F_i(X_i)\}$. Bob then outputs a set G of sets of secrets which is his guess concerning the possibilities for the set S.*

There are two kinds of strategies for Bob, namely *adaptive* and *oblivious*.

Definition 2. *In an* adaptive *strategy, Bob can design questions based on previous answers. In this case, Bob first asks question q_1 and Alice answers a_1. Then Bob asks question q_2 and Alice answers a_2 and so on.*

Definition 3. *In an* oblivious *strategy, all of Bob's question must be asked in advance of any of Alice's answers. Now, Bob asks $q_1, q_2, \ldots, q_m$ and Alice then answers all of them $a_1, a_2, \ldots, a_m$.*

In this definition, Alice corresponds to the client, his nameservers, and Akamai's authorative nameserver. Alice's secrets correspond to her nameservers' IP addresses. Bob corresponds to Akamai's webservers. The questions of an oblivious Bob are embedded as links in the original foo.com webpage and the answers are computed by Akamai's authorative nameserver upon queries from the client's nameservers. An adaptive Bob can also be implemented in this setting by embedding the link for question q_i in the page for question q_{i-1}.

We say Bob has solved the guessing secrets problem if he has learned as much as is theoretically possible concerning S. Observe that Bob can never hope to learn with certainty more than one of Alice's secrets, since Alice can always answer every question using the same X_i. We can in fact completely characterize the amount of information Bob can hope to learn about S, but first we will develop a model of this problem.

Definition 4. The Graph Model: *Let K_N denote the complete hypergraph with edge sets of size k on the set of N vertices of Ω. A set of secrets $S = \{X_1, \ldots, X_k\}$ corresponds to an edge $(X_1, \ldots, X_k)$ of K_N. Each question $F_i : \Omega \to \{0, 1\}$ induces a cut $F_i^{-1}(0)$ of K_N.*

Notice in this model, an answer $b \in \{0, 1\}$ to the question F_i implies at least one of the $X_i \in S$ are in $F_i^{-1}(b)$. Thus all edges in $F_i^{-1}(1-b)$ can be eliminated as possibilities for the set of secrets. Therefore, no matter how adversarial Alice acts, Bob can always ask questions that will reduce the resulting graph to a set of intersecting edges (i.e. a *star* or a *triangle* for $k = 2$), for if there were disjoint edges Bob could add a question which separates the edges and thereby eliminate one of them. Furthermore, Bob can not hope to learn more than this.

Now we can say formally that Bob solves the guessing secrets problem if he returns an intersecting set of edges G that contain S. We call a strategy of Bob that returns an intersecting set for any adversary a *separating* strategy, and the set of edges returned a *surviving* set.

5.3 Solutions

Clearly oblivious and adaptive separating strategies do exist. For example, Bob can simply ask the $O(N^k)$ questions corresponding to all cuts which separate one edge from the rest of the graph. However, this strategy is a bit discouraging because the information we are trying to learn is of size $O(k \log N)$. In order for our strategy to be considered *efficient*, we would like to ask just $\text{poly}(k, \log N)$ questions each of size $O(k \log N)$. Furthermore, in order to recover the surviving set from Alice's questions, Bob must spend $O(N^k)$ computation time (list all k-sets and eliminate inconsistent ones). In order for our strategy to be considered *invertible*, we would like to be able to recover the surviving set in just $\text{poly}(k, \log N)$ time.

But just how efficient a strategy can we design? In [**?**], Chung et al. proved the following theorems for the case with $k = 2$ secrets:

Theorem 1. *Let $f(N)$ be the smallest number of questions Bob must ask in an adaptive separating strategy for an initial set Ω of size N. Then,*

$$3 \log_2(N) - 5 \leq f(N) \leq 4 \log_2(N) + 3, \text{ for } N > 2$$

Theorem 2. *Let $f(N)$ be the smallest number of questions Bob must ask in an oblivious separating strategy for an initial set Ω of size N. Then,*

$$f(N) \leq (c + o(1)) \log_2 N$$

where $c = 3/\log \frac{8}{7} = 15.57\ldots$.

The proofs of the upper bounds in these theorems are either non-constructive or construct strategies that have large question sizes and not invertible. Chung et al. explicitly construct several strategies, but they are all less than optimal in some sense. Recently, Micciancio et al. [**?**] presented an adaptive strategy with the optimal $O(\log N)$ questions and an $O(\log^2 N)$ time algorithm for recovering $k = 2$ secrets.

Today, we will see a result published by Alon et al. [?] that gives an invertible oblivious strategy with the optimal $O(\log N)$ questions for $k = 2$ secrets. This strategy depends heavily on error-correcting codes. To motivate the use of error-correcting codes in problems like guessing secrets, we first present a similar problem *20 questions*, and solve it with error-correcting codes.

5.3.1 20 questions with a liar

In the 20 questions game a player, Bob, tries to discover the identity of some unknown secret drawn by a second player, Alice, from a large space of N secrets. Bob is allowed to ask Alice binary (Yes/No) questions about the secret. Alice answers each question truthfully according to her secret. The goal of Bob is to learn the secret by asking as few questions as possible. If the N possible secrets are associated with $\lceil \log N \rceil$-bit strings, then clearly $\lceil \log N \rceil$ questions are both necessary and sufficient to discover the secret. The guessing secrets problem is a generalization of this game in which Alice picks not one, but k secrets, and answers questions truthfully according to a secret of her choice.

There is another way of generalizing 20 questions game. In this generalization, Alice again has just one secret x, but she is allowed to lie, say, 10% of the time. Or, alternatively, Alice always answers truthfully, but she transmits her answers through a noisy channel which corrupts 10% of the bits. Error-correcting codes are an obvious solution to this problem. Bob can pick an appropriate code C with encoding function E and decoding function D. He asks Alice about each bit of $E(x)$. The resulting string $E(x)$ will be a codeword with at most 10% error in it. Thus, if Bob uses an error-correcting code with error recovery better than 10%, then he can recover the secret $D(E(x))$. Luckily for Bob, we know of codes that can correct strings with less than 50% error.

We would like to apply the same approach to the guessing secrets problem. We consider one secret X_1 as the "true" secret, and all other secrets as lies. We then try to apply Bob's error-correcting code strategy. However, there is a problem. Namely, even with just two secrets, Alice can lie 50% of the time if we don't pick our code carefully. Clearly, there are no error-correcting codes with 50% error recovery. But, we have a stronger assumption than in the liar game. Alice's lies must be consistent! That is, Alice's lies are just bits of $k-1$ other codewords. We will use this fact to define a class of codes, and encoding and decoding strategies for these codes.

5.3.2 Separating codes

We will concern ourselves with the 2-secret guessing problem (i.e. $k = 2$). For each secret $x \in \Omega$, we denote the sequence of answers to the questions F_i on x by $C(x) =< F_1(x), F_2(x), \ldots, F_n(x) >$. We call mapping the mapping $C : \Omega \to \{0,1\}^n$ thus defined as the code used by the strategy. There is clearly a one-one correspondence between oblivious strategies $\{F_i\}$ and such codes C (defined by $F_i(x) = C(x)_i$, where $C(x)_i$ is the i'th bit of $C(x)$). Now we can refer to a strategy using its associated code C. We say that a code C is $(2,2)$-*separating* if for every $4-tuple$ of distinct secrets $a, b, c, d \in \Omega$, there exists at least one value of i, $1 \leq i \leq n$, called the *discriminating index*, for which $C(a)_i = C(b)_i \neq C(c)_i = C(d)_i$.

Note that if Bob asks questions according to a separating code C, then for every two disjoint pairs of edges (a, b) and (c, d), Bob can rule out one of them based on the answer which Alice gives on the i'th question, where i is a discriminating index for the 4-tuple (a, b, c, d). In fact it is easy to see that the $(2, 2)$-separating property of C is also necessary for the corresponding strategy to solve the 2-secrets guessing game. Thus, there exists a $(2, 2)$-separating code $C : \Omega \to \{0, 1\}^n$ if and only if there exists an oblivious strategy for Bob to solve the 2-secret guessing problem. Now all that remains is to find good $(2, 2)$-separating codes. We use the following theorems and definitions to design such codes efficiently:

Theorem 3. *Let C be an $[n, m]_2$ binary linear code with minimum distance d and maximum distance m_1. Assume further that d, m_1 satisfy the condition $d > \frac{3m_1}{4}$. Then, C is a $(2, 2)$-separating code.*

Definition 5. *A binary linear code of block length n is an ϵ-biased code if every non-zero codeword in C has Hamming weight between $(\frac{1}{2} - \epsilon)n$ and $(\frac{1}{2} + \epsilon)n$.*

From above theorem and definition, one can easily conclude that if a binary linear code C is ϵ-biased for some $\epsilon \leq \frac{1}{14}$, then C is $(2, 2)$-separating. A simple explicit construction of ϵ-biased codes can be obtained by concatenating an outer Reed-Solomon code with relative distance $(1 - 2\epsilon)$ with an inner binary Hadamard code. However, this and other similar constructions encode our strings of length $\log N$ into codewords of length $O(\frac{\log^2 N}{\epsilon^2})$, more than the desired $O(\log N)$ length. However, in [**?**] Alon et al. proved:

Lemma 1. *For any $\epsilon > 0$, there exists an explicitly specified family of constant rate binary linear ϵ-biased codes.*

From this lemma and above discussion, we see that there is an explicit oblivious strategy for the 2-secrets guessing game that uses $O(\log N)$ questions. The problem of above method is that it is not invertible. Next we will design an invertible strategy that recovers secrets poly$(\log N)$ time.

5.3.3 List Decoding

Again, we consider the 2-secrets guessing game (i.e. $k = 2$).

Theorem 4. *Suppose that C is a $[cm, m]_2$ binary linear code which is ϵ-biased for some constant $\epsilon < \frac{1}{14}$. Suppose further that there exists a list decoding algorithm for C that corrects up to a $\frac{1}{4} + \frac{\epsilon}{2}$ fraction of errors in time $O(T(m))$. Then, C is (2,2)-separating code that gives a strategy to solve the 2-secrets guessing game for a universe size $N = 2^m$ in $O(T(\log N) + \log^3 N)$ time using $c \log N$ questions.*

Proof (Sketch): Suppose Bob uses the code $C = [n \equiv cm, m]_2$ mentioned above for his strategy. If Alice's secrets are $\{X_1, X_2\}$ and her answers to Bob are $a = (a_1, a_2, \ldots, a_n)$, then for each i, we must have either $C(X_1)_i = a_i$ or $C(X_2)_i = a_i$. As C is ϵ-biased , $C(X_1)$ and $C(X_2)$ must agree in at least $(\frac{1}{2} - \epsilon)n$ coordinates. Furthermore, since Alice doesn't lie, at least half of the $(\frac{1}{2} + \epsilon)n$ remaining coordinates must agree with one of the codewords, say $C(X_1)$. This means a is a Hamming distance of at most $(\frac{1}{4} + \frac{\epsilon}{2})n$ from $C(X_1)$.

This is the algorithm for Bob to recover the secrets:

1. Perform list decoding of the code C using the assumed algorithm to find the set

$$S = \{x \in \{0,1\}^m | \Delta(a, C(x)) \leq (\frac{1}{4} + \frac{\epsilon}{2})n\}$$

where $\Delta(x, y)$ denotes the Hamming distance between x and y.

2. For each $x \in S$, let A be the set of coordinates where $C(x)$ and a agree. Find the set of possible matching secrets S_x by erasing the coordinates A of a and using an *erasure list decoding* algorithm. As our code is a linear code, this amounts to finding all solutions to a linear system of equations. If S_x is empty (i.e. there is no x' such that Alice could have answered a with secrets (x, x')), then remove x from S. Otherwise, represent S_x by a set of basis vectors.

3. Return the set $G = \{\{x, x'\} : x \in S, x' \in S_x\}$ as the guess.

Notice a set of secrets $\{x, x'\}$ is in G if and only if it is consistent with the answer a, so in particular Alice's real secrets $\{X_1, X_2\} \in G$. Furthermore, as $\epsilon < \frac{1}{14}$, theorem **??** implies C is $(2, 2)$-separating and therefore G is an intersecting family. This shows the correctness of Bob's strategy.

The efficiency of the inversion follows as list decoding returns a constant number of solutions, so we must solve just a constant number of linear systems. Furthermore, due to our clever representation of S_x, our output is polynomial in $\log N$ even when the number of vertices in our intersecting family is $O(N)$ (e.g. the star with $(N-1)$ non-hubs obtained when $a = C(x)$ for some x). □

For detailed proof of correctness and analysis of this algorithm, see [**?**]. Also, notice if you don't believe theorem **??**, you can use the naive graph-cut approach on the constant-sized set S of vertices returned by the list decoding. If both secrets were in S, then the result will be a star or a triangle. If just one of the secrets was in S, then the result will be a star. If the result is a star, extending the star to the whole set Ω will give a correct solution G.

The only remaining problem is to prove the existence of the ϵ-biased codes used in theorem **??**. The existence of such codes is a standard result of coding theory.

Theorem 5.1. *For every positive constant $\alpha < 1/2$, the following holds. For all small enough $\epsilon > 0$, there exists an explicit asymptotically good family of binary linear ϵ-biased codes which can be list decoded up to an α fraction of errors in $O(n^2(\frac{\log(n)}{\epsilon})^O(1))$ time.*

A sketch of this result may be found in [**?**].

5.4 Open Questions

There are numerous questions about the guessing secrets problem that remain open, most of which involve some generalization or restriction of the problem. We conclude with a list some of the open questions.

1. k-Secret Guessing Problem

   In the last two sections, we analayzed strategies for $k = 2$ secrets. A natural question is to analyze the case for $k > 2$. Note that the list decoding solution no longer works because Alice's answer may be quite far from the codewords of her secrets. The hypergraph model and accompanying naive algorithm show that strategies do exist for $k > 2$. But, as stated, the naive strategy is impractical. We do have some negative results for this problem. In particular, Alon et al. showed [**?**] oblivious strategies require at least $\Omega(2^{2k} \log N)$ questions and adaptive strategies require at least $a_k \log N - a_k \log a_k$ questions where

   $$a_k = 2k - 2 + \frac{1}{2}\binom{2k-2}{k-1}.$$

   Furthermore, they present an efficient (but not invertible) oblivious strategy for this problem based on $2k$-universal families of binary strings that uses at most $ck2^{6k} \log N$ questions for some constant c independent of k.

2. Relax Solution Condition

   We can consider a restricted form of the problem in which we only require Bob to return a guess set G such that any k-set of secrets S that is consistent with Alice's answers intersects G in at least one element [**?**]. In this case, there is an efficient invertible oblivious strategy for Bob based on concatenated codes and the same list decoding ideas presented here. It may be possible to improve this result, and algorithms for adaptive strategies should be studied.

3. Non-binary Questions

   Another way to generalize our problem is to allow Bob to ask questions with more than two answers. Clearly, this makes the problem much easier for Bob. For example, in the 2-secret problem, Bob can now always determine at least one of Alice's secrets (he can eliminate triangles by asking a tertiary question). However, it is not known how non-binary questions affect the problem for $k > 2$.

4. Fixed Number of Rounds

   We can also restrict the problem by allowing Bob only a fixed number of rounds in the adaptive case. If we allow Bob to ask more than one question per round, this problem becomes a sort of hybrid between the oblivious and adaptive strategies. How can we combine the solutions for oblivious and adaptive strategies to get an optimal strategy in this case?

5. Lying

   Finally, we can allow Alice to lie some fraction of the time. That is, for say 10% of the questions, Alice can give any answer she chooses, independent of her secrets. Can we find codes such that this new power doesn't help Alice too much? How about in the adaptive case?

Bibliography

[1] N. Alon, J. Bruck, J. Naor, M. Naor, and R. Roth. *Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs.*, IEEE Trans. Info. Thy., 38(2):509–512, March 1992.

[2] Fan R. K. Chung and Ronald Graham and Frank Thomson Leighton, *Guessing Secrets*, Symposium on Discrete Algorithms, p723-726, 2001.

[3] Noga Alon, Venkatesan Guruswami, Tali Kaufman, Madhu Sudan, *Guessing Secrets Efficiently via List Decoding*, SODA 2002.

[4] Daniele Micciancio, and Nathan Segerlind. *Using prefixes to efficiently guess two secrets.*, Manuscript, July 2001.