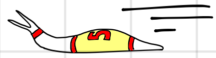
CS250/EE387 - Lecture 17 - Locality and Lists

Agenda

1. Motivation: Learning Boolean Fns
2. Goldreich-Levin Algorithm
3. Local List Decoding.

Gastropod Fact

The town of Elma, WA has an annual Slug Fest featuring a "world's fastest slug" race, on a course 24in long.
The average banana slug can move about 6.5in per minute.



0 Recap.

Last time we talked about LOCALLY CORRECTABLE CODES.

The basic principle was illustrated by the Hadamard Code $RM_2(m,1)$,

$$RM_2(m,1) = \left\{ \left(\langle \omega, \alpha_1 \rangle, \ldots, \langle \omega, \alpha_{2^m} \rangle \right) : \omega \in \mathbb{F}_2^m \right\}$$

The key was that $\forall i$, $\langle \omega, \alpha_i + \beta \rangle + \langle \omega, \beta \rangle = \langle \omega, \alpha_i \rangle$, so to locally recover $\langle \omega, \alpha_i \rangle$ we query $\langle \omega, \alpha_i + \beta \rangle, \langle \omega, \beta \rangle$, HOPE they are not corrupted, and add them together.

1 Motivation: Learning Boolean Fns.

Throughout this lecture, capital letters mean the range is $\{\pm 1\}$.
Lowercase $g: \mathbb{F}_2^n \to \mathbb{F}_2$ have range $\mathbb{F}_2$.

Suppose you have some Boolean function $G: \mathbb{F}_2^m \to \{-1,+1\}$.
You have query access to G and you'd like to learn an approximation $\hat{G}$ to G.

Def: For $G: \mathbb{F}_2^m \to \{-1,1\}^m$, the FOURIER TRANSFORM of G over $\mathbb{F}_2$ is $\hat{G}: \mathbb{F}_2^m \to \mathbb{R}$ given by

$$\hat{G}(\omega) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^m} G(x) \cdot (-1)^{\langle x, \omega \rangle}$$

If you haven't seen this before, but have seen the Fourier Transform over $\mathbb{C}$, all the same things hold. In particular:

$$G(x) = \sum_{\omega \in \mathbb{F}_2^m} \hat{G}(\omega) (-1)^{\langle x, \omega \rangle}$$

and:

"Parseval's Thm"

$$1 = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^m} G(x)^2 = \sum_{\omega \in \mathbb{F}_2^m} |\hat{G}(\omega)|^2$$

so in particular the number of Fourier coefficients $\hat{G}(\omega)$ so that $\hat{G}(\omega) > \tau$ is $\leq 1/\tau^2$.

Suppose we want to learn G from samples.

If the Fourier spectrum of G is "spiky," it suffices to estimate $y_\omega \approx \hat{G}(\omega)$ for all ω so that $|\hat{G}(\omega)| > \tau$.

Indeed, then we'd have

$$G(x) \approx \sum_{\omega : |\hat{G}(\omega)| > \tau} \hat{G}(\omega) (-1)^{\langle x, \omega \rangle} \approx \sum_{\omega : |\hat{G}(\omega)| > \tau} y_\omega \cdot (-1)^{\langle x, \omega \rangle}.$$

Turns out, we can estimate any <u>particular</u> $\hat{G}(\omega)$ from samples:

$$\hat{G}(\omega) := \frac{1}{2^m} \sum_x G(x) (-1)^{\langle x, \omega \rangle},$$ so choose a bunch of x's at random, and estimate the sum.

But we can't do this for all 2^m coeffs $\hat{G}(\omega)$, or else that takes $\Omega(2^m)$ samples – kinda dumb.

Instead we'll just do it for the big ones... but we need to know which those are.

GOAL. Given query access to $G(x)$ and a parameter $\tau > 0$, find a set S of size poly(m) so that $\forall \omega$ w/ $|\hat{G}(\omega)| \geq \tau$, $\omega \in S$.

NOTE: We'll lose the $|\cdot|$ in the GOAL for simplicity. By repeating whatever we come up with for $-G$, it will be fine.

Now, $\hat{G}(\omega) \geq \tau$

remember, $\in \{\pm 1\}$

$$\iff \frac{1}{2^m} \sum_{x \in \mathbb{F}_2^m} G(x) \cdot (-1)^{\langle x, \omega \rangle} \geq \tau$$

$$\iff \frac{1}{2^m} \left(\left|\{x : G(x) = (-1)^{\langle x, \omega \rangle}\}\right| - \left|\{x : G(x) \neq (-1)^{\langle x, \omega \rangle}\}\right| \right) \geq \tau$$

$$\iff \frac{1}{2^m} \left(2 \left|\{x : G(x) = (-1)^{\langle x, \omega \rangle}\}\right| - 1 \right) \geq \tau$$

$$\iff \frac{1}{2^m} \left|\{x : G(x) = (-1)^{\langle x, \omega \rangle}\}\right| \geq \frac{1}{2} + \frac{\tau}{2}$$

$$\iff \frac{1}{2^m} \left|\{x : g(x) = \langle x, \omega \rangle\}\right| \geq \frac{1}{2} + \tau/2$$ where $G(x) = (-1)^{g(x)}$, aka, $g(x) = \begin{cases} 0 & G(x) = 1 \\ 1 & G(x) = -1 \end{cases}$

$$\iff \delta(g, l_\omega) \leq \frac{1}{2} - \tau/2,$$ where $l_\omega(x) = \langle x, \omega \rangle$ and $(l_\omega(x_1), l_\omega(x_2), \ldots, l_\omega(x_{2^n}))$ is a Hadamard codeword!

NEW GOAL. Given query access to a received word $g:\mathbb{F}_2^m \to \mathbb{F}_2$, find all the Hadamard codewords $(\langle w, x_1\rangle, \dots, \langle w, x_{2^m}\rangle) = (\ell_w(x_1), \dots, \ell_w(x_{2^m}))$ so that $\delta(g, \ell_w) \le \frac{1}{2} - \varepsilon$.

That is, we'd like to LIST DECODE the Hadamard Code... in SUBLINEAR TIME!

NOTICE: Dist(Hadamard Code) $= \frac{1}{2}$, so we can only uniquely decode up to radius $1/4$.

(relative)

You showed this on HW1.
Also, $\mathrm{dist}(RM_2(m,r)) = \frac{1}{2^r}$,
so $\mathrm{dist}(RM_2(m,1)) = 1/2$.

But we could hope to list-decode up to $1/2$. In this case, the Johnson radius is $J_2(\frac{1}{2}) = \frac{1}{2}\left(1 - \sqrt{1 - 2\cdot\frac{1}{2}}\right) = \frac{1}{2}$, so we know that the list size isn't too big.

We also know this from the argument with Parseval's Thm earlier.

(2) GOLDREICH-LEVIN ALG.

To warm up, let's do it for $\frac{1}{4}$:

ALG 0.

Input: query access to $g:\mathbb{F}_2^m \to \mathbb{F}_2$, a parameter ε.

Output: The $w \in \mathbb{F}_2^m$ s.t. $\delta(g, \ell_w) \le \frac{1}{4} - \varepsilon$, w/ prob $\ge 99/100$.

Draw $\beta_1, \dots, \beta_T \in \mathbb{F}_2^m$ uniformly at random.

For $i = 1, \dots, m$:

Set $T = O(m/\varepsilon^2)$

For $t \in 1, \dots, T$:

Set $\tilde{w}_i(\beta_t) = g(e_i + \beta) + g(\beta)$

$\tilde{w}_i \leftarrow \mathrm{MAJ}(\tilde{w}_i(\beta_t))$

RETURN $\tilde{w} = (\tilde{w}_1, \tilde{w}_2, \dots, \tilde{w}_m)$

Notice this alg makes $T(1+m)$ queries: $g(\beta_t)$ for $t = 1, \dots, T$
$g(\beta_t + e_i)$ for $t \in [T]$, $i \in [m]$

At this point we've seen this alg several times.

Why does this work? As we've seen before:

$$\mathbb{P}\{\tilde{\omega}_i(\beta) \text{ is incorrect}\} \leq \mathbb{P}\{\text{either } g(e_i+\beta) \text{ or } g(\beta) \text{ were in error}\}$$
$$\leq \left(\tfrac{1}{4}-\epsilon\right) + \left(\tfrac{1}{4}-\epsilon\right)$$
$$= \tfrac{1}{2} - 2\varepsilon.$$

$$\mathbb{P}\{\text{More than } \tfrac{1}{2} \text{ of the } \tilde{\omega}_i(\beta) \text{ are incorrect}\}$$

$$= \mathbb{P}\left\{ \frac{1}{T}\sum_{t=1}^{T}\left[\mathbb{1}\{\tilde{\omega}_i(\beta) \text{ incorrect}\} - \left(\tfrac{1}{2}-2\varepsilon\right)\right] > 2\varepsilon \right\}$$

$$\leq \frac{\frac{1}{T^2}\sum_{t=1}^{T} \mathbb{E}\left(\mathbb{1}\{\tilde{\omega}_i(\beta) \text{ incorrect}\} - \left(\tfrac{1}{2}-2\varepsilon\right)\right)^2}{(2\varepsilon)^2} \qquad \text{by Chebyshev}$$

$$= \frac{1}{T\cdot 4\varepsilon^2} \cdot \left(\tfrac{1}{2}-2\epsilon\right)\left(\tfrac{1}{2}+2\epsilon\right)$$

$$= \frac{(1-16\varepsilon^2)}{T\cdot 16\cdot\varepsilon^2}$$

$$\leq 1/100m \text{ if we choose } T = \Theta(m/\varepsilon^2).$$

Now union bound over all i and win.

OK, but now we want to do it up to $\frac{1}{2}-\varepsilon$, not $\frac{1}{4}-\varepsilon$.

Suppose we had access to a magic genie who will just tell us the correct value $\langle \omega, \beta_j \rangle$
But we can only ask the genie for T values.

ALG 1.

Input: query access to $g: \mathbb{F}_2^m \to \mathbb{F}_2$, a parameter ε, and a magic genie.
Output: An $\omega \in \mathbb{F}_2^m$ s.t. $\delta(g, \ell_\omega) \leq \frac{1}{2}-\epsilon$, w/ prob 99/100.

set $T = O(m/\epsilon^2)$

Draw $\beta_1, \ldots, \beta_T$ uniformly at random.
Ask the genie for $b_1, \ldots, b_T$ so that $b_i = \langle \omega, \beta_i \rangle$
For each $i = 1, \ldots, m$:
- For $t \in 1, \ldots, T$:
  - Set $\tilde{\omega}_i(\beta_t) = g(e_i + \beta_t) + b_t$
- $\tilde{\omega}_i \leftarrow \mathrm{MAJ}(\tilde{\omega}_i(\beta_t))$

RETURN $\tilde{\omega} = (\tilde{\omega}_1, \tilde{\omega}_2, \ldots, \tilde{\omega}_m)$

This alg makes $T \cdot m$ queries.

Now, the same argument works:

$$\mathbb{P}\{\tilde{\omega}_i(\beta_t) \text{ is incorrect}\} = \mathbb{P}\{g(e_i+\beta_t) \text{ incorrect or the genie lied}\}$$

$$= \mathbb{P}\{g(e_i+\beta_t) \text{ incorrect}\} \quad \text{(because genies don't lie).}$$

$$\leq \frac{1}{2}-\varepsilon,$$

so everything goes through as before.

The problem: WE DON'T HAVE A GENIE.

ALG 2.

Input: query access to $g : \mathbb{F}_2^m \to \mathbb{F}_2$, a parameter ε,
Output: A list of $\omega \in \mathbb{F}_2^m$ s.t. $\delta(g, \ell_\omega) \leq \frac{1}{2} - \epsilon$, w/ prob 99/100.

Initialize $S \leftarrow \phi$
For each $(b_1, \dots, b_T) \in \mathbb{F}_2^T$:

define $\mathrm{GENIE}_{b_1, \dots, b_T}(t) = b_t$

Run ALG 1. using this genie to obtain ω
Add ω to S.

RETURN S

Why is this a good idea?

- If $\delta(\ell_\omega, g) \leq \frac{1}{2} - \epsilon$, then $\exists\ b_1, \dots, b_T$ $(= \langle \omega, \beta_1 \rangle, \dots, \langle \omega, \beta_T \rangle)$ so that ALG1 returns ω. Thus ω ends up in the list S.

Why is this a bad idea?

- $|S| = 2^T = 2^{O(m/\epsilon^2)} \geq |\mathbb{F}_2^m|$.
- But $S \subseteq \mathbb{F}_2^m$ was supposed to be a small subset.

To fix this, we will use a PSEUDORANDOM genie.

To see what this means, consider the following way of picking the β's.

- Choose $\beta_1, \dots, \beta_\ell$ randomly [and let $\ell = \log(T)$]
- For $A \subseteq [\ell]$, define $\beta_A := \sum_{i \in A} \beta_i$
- Now I have $2^\ell = T$ different values of β.
- CLAIM. $\{\beta_A : A \subseteq [\ell]\}$ are PAIRWISE INDEPENDENT.
  aka, for any $A \neq A'$, β_A and $\beta_{A'}$ are independent.

proof.

pf. $\beta_A - \beta_{A'} \overset{\text{in } \mathbb{F}_2}{=} \beta_A + \beta_{A'} = \sum_{t \in A \Delta A'} \beta_t$

symmetric difference, which is nonempty

So, conditioning on $\beta_{A'}$, we have $\beta_A = \underbrace{\beta_{A'}}_{\text{fixed}} + \underbrace{\sum_{t \in A \Delta A'} \beta_t}_{\text{uniformly random in } \mathbb{F}_2}$,
hence $\mathbb{P}\{\beta_A = 1 \mid \beta_{A'}\} = \frac{1}{2}$,
and so they are independent.

- Notice that our correctness argument before never used the fact that the β_i were fully independent: for Chebyshev we only needed pairwise independence.
- So ALG1. works just fine with these β's!

ALG 3.

Input: query access to $g : \mathbb{F}_2^m \to \mathbb{F}_2$, a parameter ε, and a magic genie.
Output: An $\omega \in \mathbb{F}_2^m$ s.t. $\delta(g, \ell_\omega) \leq \frac{1}{2} - \epsilon$, w/ prob $99/100$.

Draw $\beta_1, \ldots, \beta_\ell$ uniformly at random, $\leftarrow \ell = \log(m/\epsilon^2) + O(1)$
Ask the genie for $b_1, \ldots, b_T$ so that $b_i = \langle \omega, \beta_i \rangle$.

For $A \subseteq [\ell]$, let $\beta_A = \sum_{t \in A} \beta_t$, let $b_A = \sum_{t \in A} b_t$.

FOR each $i = 1, \ldots, m$:
  For $A \subseteq [\ell]$:
    Set $\tilde{\omega}_i(\beta_A) = g(e_i + \beta_A) + b_A$
  $\tilde{\omega}_i \leftarrow \mathrm{MAJ}(\tilde{\omega}_i(\beta_A))$

RETURN $\tilde{\omega} = (\tilde{\omega}_1, \tilde{\omega}_2, \ldots, \tilde{\omega}_m)$

This alg makes $T \cdot m$ queries.

Notice that if the genie is correct about $b_1, \ldots, b_\ell$, then $\langle \omega, \beta_A \rangle = \sum_{t \in A} \langle \omega, \beta_t \rangle = \sum_{t \in A} b_t = b_A$, so the genie is correct about b_A $\forall A \subseteq [\ell]$.

This alg. is correct for exactly the same reason as before, since the β_A are pairwise independent.

ALG 4 (GOLDREICH-LEVIN)

Input: query access to $g : \mathbb{F}_2^m \to \mathbb{F}_2$, a parameter ε,
Output: A list of $w \in \mathbb{F}_2^m$ s.t. $\delta(g, \ell_w) \le \frac{1}{2} - \epsilon$, w/ prob 99/100.

Initialize $S \leftarrow \phi$
For each $(b_1, \dots, b_\ell) \in \mathbb{F}_2^\ell$: set $\ell = \log(m/\epsilon^2) + O(1)$

define $\mathrm{GENIE}_{b_1,\dots,b_\ell}(t) = b_t$

Run ALG 3 using this genie to obtain ω
Add ω to S.

RETURN S

We have basically already proven:

THM. The Goldreich Levin algorithm makes $\mathrm{poly}(m/\varepsilon)$ queries to g and returns a list $S \subseteq \mathbb{F}_2^m$ of size at most $\mathrm{poly}(m/\varepsilon)$ so that, $\forall w \in \mathbb{F}_2^m$ with $\delta(\ell_w, g) \le \frac{1}{2} - \epsilon$, $w \in S$.

Informal COR. (KUSHILEVITZ-MANSOUR)

If $G : \mathbb{F}_2^m \to \{\pm 1\}$ is a Boolean function, then we can estimate

$$\tilde{G}(x) \approx \sum_{\omega : |\hat{G}(\omega)| > \tau} \hat{G}(\omega) \cdot (-1)^{\langle x, \omega \rangle}$$

using $\mathrm{poly}(m/\tau)$ queries.

3. LOCAL LIST DECODING.

What we just saw was a LOCAL LIST DECODING ALGORITHM.

DEF. $C \subseteq \Sigma^n$ is (Q, ρ, L) - LOCALLY LIST DECODABLE if:

There is a randomized algorithm $\mathcal{A}$ that outputs at most L other algs $\mathcal{B}_1, \ldots, \mathcal{B}_L$ so that:

- $\forall i \in [L]$, $\mathcal{B}_i$ takes an input $j \in [n]$, uses at most Q queries to $g \in \Sigma^n$.
- $\forall g \in \Sigma^n$,

$$\forall c \in C \text{ w/ } \delta(c, g) \le \rho, \ \exists i \text{ s.t. } \forall j \in [n]:$$

$$\mathbb{P}\{ \mathcal{B}_i(j, \text{access to } g) = c_j \} \ge \frac{2}{3}$$

Think of each $\mathcal{B}_i$ as a different genie.
In the previous example, the $\mathcal{B}$'s were indexed by $(b_1, b_2, \ldots, b_\ell) \in \mathbb{F}_2^\ell$:

GENIE $\mathcal{B}_{(b_1, b_2, \ldots, b_\ell)}$ (query access to g, eval pt α):

$\ell \leftarrow \log(1/\varepsilon^2) + O(1)$
For $A \subseteq [\ell]$:
$\quad \tilde{w}_\alpha(\beta_A) = g(\alpha + \beta_A) + \sum_{j \in A} b_j$
$\tilde{w}_\alpha \leftarrow \mathrm{MAJ}(\tilde{w}_\alpha(\beta_A) : A \subseteq [\ell])$
RETURN $\tilde{w}_\alpha$

NOTE: This is not quite the same as in our Goldreich–Levin version, since that was supposed to recover all of w, and this just guesses $\langle w, \alpha \rangle$. But the idea is the same.

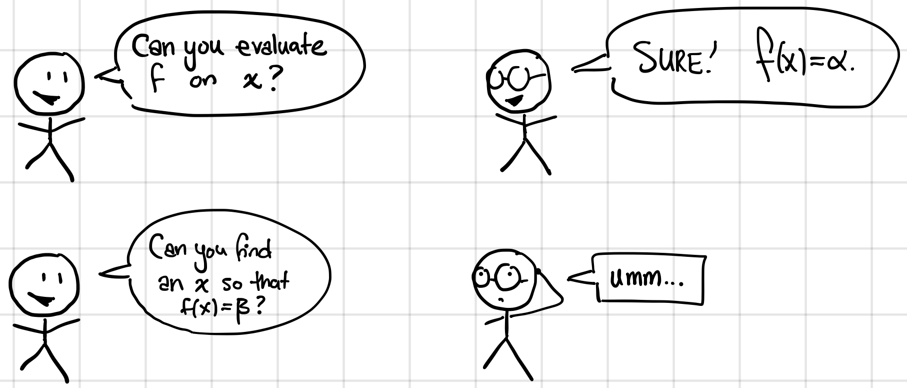
The reason we bother to give LOCAL LIST DECODING a name is because it has many applications. We've already seen one in learning theory, and here's another:

4 PRGs from OWFs

(This is what Goldreich + Levin were interested in).

WARNING: This will be extra handwavey.

"DEF." A ONE-WAY FUNCTION (OWF) is a function that is easy to apply by hard to invert.



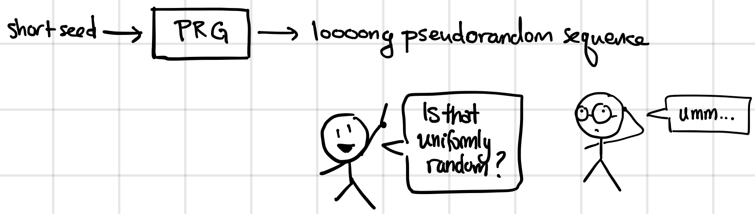
- We don't know if OWFs exist. In fact, $\exists$ OWF $\Rightarrow$ P $\neq$ NP.

Intuitively, a OWF gives a problem that's hard to solve but easy to check, and that's what P $\neq$ NP means.

- But there are several candidates: factoring, discrete log, etc.
- And if a OWF exists, we can do some cool things with it.

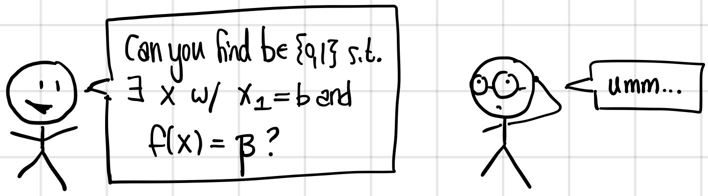
"DEF" PSEUDORANDOM GENERATOR.

A PRG has output that is not very random, but is computationally difficult to distinguish from uniform.



We might try to make a PRG from a OWF as follows:

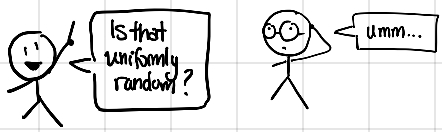
- Say f is a OWF, $f: \mathbb{F}_2^k \to \mathbb{F}_2^k$ ← Technically, f should be a ONE-WAY PERMUTATION.
- SUPPOSE that this also means that it's hard to guess x_1 given $f(x)$. (*)



- Now consider the PRG:

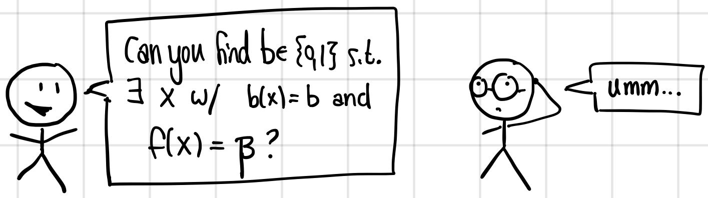
$$x \longrightarrow \boxed{\text{PRG}} \longrightarrow \left(x_1,\ [f(x)]_1,\ [f(f(x))]_1,\ [f(f(f(x)))]_1,\ \dots\right)$$

Random seed



- Turns out this is a good PRG, assuming (*).
- But there is no reason (*) should be true.

"DEF" A HARDCORE PREDICATE $b(x)$ for $f(x)$ is a function $b: \mathbb{F}_2^k \to \mathbb{F}_2$ so that it's hard to guess $b(x)$ given $f(x)$.



So in order to get PRGs from OWFs, we want a hardcore predicate for our OWF f.

In fact, we get this from the local list-decodability of the Hadamard code.

"CLAIM." Let $f : \mathbb{F}_2^m \to \mathbb{F}_2^m$ be a ONE-WAY PERMUTATION.
Then it's hard to guess $\langle \alpha, x \rangle$ given $f(x)$ and α.

aka, for all $\alpha \in \mathbb{F}_2^m$, $\langle \alpha, x \rangle$ is a hardcore predicate for $\tilde{f} : (x, \alpha) \mapsto (f(x), \alpha)$.

"pf." Suppose there were some alg Q so that

$$\mathbb{P}_{\alpha}\left\{ Q(\alpha, f(x)) = \langle \alpha, x \rangle \right\} \geq \frac{1}{2} + \epsilon .$$

Aka, Q has just a slight advantage.

Then I can get query access to $g(\alpha) := Q(\alpha, f(x))$, which is a very noisy version of a Hadamard codeword.

Now I can use my local list-decoding algorithm to obtain a list $\mathcal{L}$ of $O(1/\epsilon^2)$ possible x's.

Then I compute $\{ f(x) : x \in \mathcal{L} \}$, find x s.t. $f(x) = \beta$, and return it.

So f is easy to invert after all!

QUESTIONS to PONDER

1. Can you locally list decode $RM_q(m, r)$ for $r < q$?
2. Can you learn Fourier-sparse fns from $\mathrm{poly}(m/\epsilon)$ RANDOM queries?
3. Can you think of other applications of local list decoding?