6.893: Algorithms and Signal Processing

Lecture 5:
O(k log n)-time algorithm
(for worst case inputs)
Basic block

• Assume:
  – $n$ is a power of 2
  – $\hat{a}$ contains at most $k$ non-zero coefficients with “polynomial precision”
    • e.g., $\hat{a}_u$ in $\{-n^{O(1)}, \ldots, n^{O(1)}\}$

• Then there is an $O(k \log n)$-time algorithm that
  – Outputs at most $k$ potential coefficients
  – Each actual non-zero coefficient of $\hat{a}$ is output correctly with probability $1-\beta$ for some constant $\beta > 0$
Basic block – flat window function

- Pick \( g \) s.t.
  - \( \varepsilon', \varepsilon = \Theta(1/k) \)
  - \( \varepsilon' = (1-\gamma)\varepsilon \)
  - \( \delta = 1/n^{O(1)} \)

- Then
  \[ |\text{supp}(g)| = O(k \log(n)/\gamma) \]
Basic block - algorithm

• Set $B=1/(2\varepsilon)$
  – We will take $B$ samples
• Partial-Recovery($B, \gamma$)
  – Choose random $\sigma, \beta$
  – Define $a'_j = a_{\sigma j} \omega^{-j} \beta$ (so that $\hat{a}'_u = \hat{a}_u^{1/\sigma (u+\beta)}$)
  – Define $a''_j = a'_{j+1}$ (so that $\hat{a}''_u = \hat{a}'_u \omega^u$)
  – Compute $\hat{c}'_{jn/B}, j=0..B-1$, where $\hat{c}' = \hat{a}'*\hat{g}$
    (how? - see next slide)
  – Compute $\hat{c}''_{jn/B}, j=0..B-1$, where $\hat{c}'' = \hat{a}''*\hat{g}$
    (*)
  – For $j=0$ to $B-1$
    • If $|\hat{c}'_{jn/B}|>1/2$ then
      – $val = \hat{c}'_{jn/B}$
      – $pos = 1/\sigma \text{ (round(phase(} \hat{c}''_{jn/B}/\hat{c}'_{jn/B} ))+\beta)$
      – Add ($val$, $pos$) to the output list
Computing \( \hat{c}_{jn/B} \), \( j=0..B-1 \)

- **Option 1 direct:** Compute the DFT of
  \[
  (a \times g)_{-w/2} \ldots (a \times g)_{w/2}
  \]
  \( w=O(k \log(n)/\gamma) \)
  - This follows directly the outline
  - However, it runs in \( O(k \log(n)/\gamma) \times \log n \) time
  - ...and it computes too many samples of \( \hat{a} \times \hat{g} \) anyway

- **Option 2:** alias \( a \times g \) into \( B \) bins first. I.e.,
  - Compute \( b_i = \sum_t (a \times g)_{i+t \cdot B} \)
  - Compute the DFT of \( b_0 \ldots b_{B-1} \)
Basic block - analysis

- Claim 1: At most $k$ entries in $\hat{c}_0 \ldots \hat{c}_{B-1}$ have magnitudes >1/2, and therefore are reported
- Claim 2: For each $u$ in $\text{supp}(\hat{a})$, the probability $u$ is not correctly reported is at most $O(k\varepsilon + \gamma)$
- Proof:
  - Probability of being mapped within $O(\varepsilon n)$ of some other coefficient is $O(k\varepsilon)$
    (Lemma from previous lecture)
  - Probability that the coefficient is not sampled properly is at most $O(\gamma)$
    (since the position of the center of the “flat peak” is chosen uniformly at random)
Full algorithm

- **Modification:** Partial-Recovery(B, γ, List)
  - Insert the following code in (*)
    - For each (val,pos) in List
      - $u=\sigma \text{pos} - \beta$
      - let $j$ be the closest bin to $u$
      - $\text{off}=u-j \ n/B$
      - $c'_{jn/B} = c'_{jn/B} - \text{val} \ \hat{g}_{\text{off}}$
      - $c''_{jn/B} = c''_{jn/B} - \text{val} \ \omega^u \ \hat{g}_{\text{off}}$

- **Main algorithm:**
  - List=$\phi$
  - For $t=1$ to log $k$
    - $B_t=Ck/4^t$, $\gamma_t=1/(C2^t)$ (this defines $\varepsilon_t$ and $\varepsilon'_t$)
    - List=List + Partial-Recovery($B_t$, $\gamma_t$, List)
Full algorithm - analysis

• Let $\hat{e}_t$ be the contents of List at the end of stage $t$
• Define a “good” event $E_t$:
  \[ r_t = ||\hat{a}_t - \hat{e}_t||_0 < k/8^t \]
• Conditioned on $E_{t-1}$, for any $u$, the probability of failure to recover $u$ is at most the sum of:
  - $r_{t-1} \varepsilon_t = O(k/8^{t-1} 1/(Ck/4^t)) = O(1)/C 1/2^t$
  - $\gamma_t = 1/(C2^{t-1})$
• Therefore:
  \[ \Pr[\hat{E}_t|E_{t-1}] < \Pr[\text{fraction of failures} > 1/(3*8)] < O(1)/C 2^{-t} \]
  and
  \[ \Pr[\hat{E}_1 v \hat{E}_2 v ... v \hat{E}_{\log k}] < O(1)/C (1/2+1/4+1/8+...)=O(1)/C \]
Running time

• For $t=1$ to $\log k$
  – $B_t = Ck/4^t$, $\gamma_t = k/(C2^t)$ (this defines $\varepsilon_t$ and $\varepsilon'_t$)
  – List = List + Partial-Recovery($B_t$, $\gamma_t$, List)

• Time:
  – DFT: $O(k \log n) + O(k/4 \log n) + ...$
  – List update: $O(\log n) \times k$