Sublinear Algorithms

Lecture 09: Applications I
Previous 2 lectures:
Property Testing

Next 3 lectures:
Technology transfer from Sublinear Algorithms to
Traditional algorithms

This lecture:
Sparse Convolution
Polynomial Multiplication and Convolution

\[ \alpha(x) = \sum_{i=0}^{d-1} \alpha_i x^i \quad \beta(x) = \sum_{i=0}^{d-1} \beta_i x^i \]

\[ \alpha(x) \cdot \beta(x) =? \]

can be computed with FFT in time \( O(d \log d) \)

\( (1 + x + x^3) \cdot (101 + x^2 + x^5) \) \( (1 + x + x^{1001}) \cdot (-1 + x^4 + x^{10002}) \)

**r-th coefficient:** \( \sum_{i=0}^{r} \alpha_i \cdot \beta_{r-i} \)

Convolutions

\( u, v \in \mathbb{R}^d \)

\( u \ast v \in \mathbb{R}^{2d-1} \quad (u \ast v)_r = \sum_{j=0}^{r} u_j v_{r-i} \)

Convolution \sim \text{polynomial multiplication}
Some applications of convolution

\[ A, B \subseteq \mathbb{Z}, \text{ compute } A + B = \{a + b | a \in A, b \in B\} \]

\[ \left( \sum_{a \in A} x^a \right) \cdot \left( \sum_{b \in B} x^b \right) \]

**Subset Sum**: Given set \( X = \{x_1, x_2, \ldots, x_n\} \) and a target \( t \), does

\[ t \in \{0, x_1\} + \{0, x_2\} + \ldots + \{0, x_n\}? \]

**Knapsack (via 2D convolution)**: Given tuples \((v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)\)

And a budget \( W \), among the points in

\[ \{0, (v_1, w_1)\} + \{0, (v_2, w_2)\} + \ldots + \{0, (v_n, w_n)\} \]

*with ordinate at most \( W \)*

which is the one with the largest abscissa?

**Pattern Matching**: Can be written as a convolution

between text and pattern
Sparse Convolution

Compute convolution in time near-linear to the size of

\[ \text{supp}(u \star v) = \{i : (u \star v)_i \neq 0\} \]

Going beyond \( O(d \log d) \)

- Boolean Convolution
- Nonnegative Convolution
- Most general case

An intermediate notion: cyclic convolution

\[ u \star_c v \in \mathbb{R}^d \]

\[ (u \star_c v)_i = \sum_{j=0}^{d-1} u_j v_{(i-j) \mod d} \]
Hashing the support

**Approach:** Get our hands on the support of the convolution by performing a \(<<d\)-length convolution

(Intermediate) Promise Problem: Solve sparse convolution, given a set \(T\) which contains the support.

**Folding:** Pick a number \(B\), and fold the vectors:

\[
\begin{align*}
\tilde{v}, \tilde{u} &\in \mathbb{R}^B \\
\tilde{v}_\ell &= \sum_{i \in [d] : i \mod B = \ell} v_i \\
\tilde{u}_m &= \sum_{i \in [d] : i \mod B = \ell} u_i
\end{align*}
\]

Claim: \(\tilde{v} \ast \epsilon \tilde{u}\) Resembles a hashing of \(u \ast v\) to \(B\) buckets.

Short: sum up every \(B\)-th entry to obtain a \(B\)-length vector
Hashing the support

**Approach:** Get our hands on the support of the convolution by performing a \(<d\)-length convolution

(Intermediate) Promise Problem: Solve sparse convolution, given a set \(T\) which contains the support.

**Hashing claim:**

\[(\tilde{u} \ast_c \tilde{v})_r = \sum_{i \in [d]: i \mod B = r} (u \ast v)_i\]

In which term does \(u_j v_{i-j}\) contribute to?

\(j + (i - j) = i\) It contributes to the \(i\)-th term in the unfolded version

\((j, i - j) \mapsto (j \mod B, (i - j) \mod B)\)

\((j \mod B + (i - j) \mod B) \mod B = i \mod B\)

It contributes to the \((i \mod B)\)-th term in the folded version
Hashing the support

(Intermediate) Promise Problem: Solve sparse convolution, given a set $T$ which contains the support.

$$(\tilde{u} * c \tilde{v})_r = \sum_{i \in [d]: i \mod B = r} (u * v)_i$$

If all $i$ in $T$ are distinct mod $B$ (hashed to distinct bucket) we can recover the initial convolution from $B$ buckets.

How? For every $i$ in $T$, look at the bucket $i \mod B$, and read its value.

Ensure distinctness: Pick random prime $B$ of size $\sim |T|$ $(\log d)^2$

Bad event: $i$ in $T$ collides with some $j$ in $T$, i.e. $i \mod B = j \mod B$

$$\Pr \{i \mod B = j \mod B\} = \Pr \{B|i - j\} = \frac{\text{no. divisors of } i - j}{\text{no. primes}} \approx 2|T| \log^2 d \leq \frac{\log d}{2|T| \log d} = \frac{1}{2|T|}$$

$$\Pr \{i \text{ collides with some } j \in T\} \leq \frac{|T|}{2|T|} = \frac{1}{2}$$

Repeat $O(\log |T|)$ times – and union bound!
So, where are we?

(Intermediate) Promise Problem: Sparse convolution,
    given a set $T$ which contains the support,
    Can be solved in time $O(|T| (\log d)^4)$

Removing this assumption in the nonnegative case

**Theorem:** Given two vectors, $v, w$ we can compute their convolution
    in time $O(\text{out} \ (\log d)^5)$, where $\text{out}$ is
the number of non-zero coordinates in the convolution

Let $d$ be a power of 2. We shall discuss the case
    of finding $A + B = \{a+b \mid a \in A, b \in B\}$
    in time $O(|A+B| (\log d)^5)$, for $A, B$ in $[d]$.

\[
A' = \left\{ \left\lfloor \frac{a}{2} \right\rfloor \mid a \in A \right\} \\
B' = \left\{ \left\lfloor \frac{b}{2} \right\rfloor \mid b \in B \right\}
\]

\[
\begin{align*}
T & := 2 \cdot (A' + B') + \{0, 1, 2\} \\
\text{Compute } A' + B' & \text{ recursively} \\
\text{Compute } A + B & \text{ using the (intermediate) Promise problem routine}
\end{align*}
\]
Sketch of the general case

(Intermediate) Promise Problem still holds for general vectors, but the recursion fails

But can still solve the problem in near-linear time in \((\text{size of input}) + (\text{size of output})\)

Idea: Before folding, add an identifier.
Let \(\omega\) be a \((2d)\)-th root of unity.

\[
\tilde{v}_m = \sum_{i \in [d]: i \mod B = m} v_i \cdot \omega^i
\]
\[
(\tilde{u} \cdot \tilde{v})_r = \sum_{i \in [d]: i \mod B = r} (u \cdot v)_i \cdot \omega^i
\]
\[
\tilde{u}_\ell = \sum_{i \in [d]: i \mod B = \ell} u_i \cdot \omega^i
\]

If \(i\) is isolated, then from the complex part of \(u\cdot\), we can read \(\omega^i\)
From \(\omega^i\), we can learn \(i\) in \(O(\log d)\) time.

Iterate over all \(B\) buckets, and find possible locations

If we knew \(\text{out} = \text{size of output}\), we’d set \(B = 10\text{out} (\log d)^2\)

We recover a constant fraction of coordinates + introduce false positives
... so “recurse” to clean the errors.
Recap of this lecture

Sparse Convolutions:
Compute convolutions faster than FFT

... with the help of some hashing/sketching

We can compute convolutions in near-linear output-sensitive time

Next lecture by Karl Bringmann
Thank you