UNIVERSITÄT DES SAARLANDES

max planck institut informatik

Sublinear Algorithms

Lecture 09: Applications I

European Research Council

Established by the European Commission

Previous 2 lectures:

Property Testing

Next 3 lectures:

Technology transfer from Sublinear Algorithms to *Traditional* algorithms

This lecture:

Sparse Convolution

Polynomial Multiplication and Convolution

$$\alpha(x) = \sum_{i=0}^{d-1} \alpha_i x^i \qquad \beta(x) = \sum_{i=0}^{d-1} \beta_i x^i$$

$$\alpha(x) \cdot \beta(x) = ?$$

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: $\sum_{i=0}^{r} \alpha_i \cdot \beta_{r-i}$

<u>Convolution</u>

$$u, v \in \mathbb{R}^d$$

$$u \star v \in \mathbb{R}^{2d-1} \;\; (u \star v)_r = \sum_{j=0}^{r} u_i v_{r-i}$$

Convolution ~ polynomial multiplication

Some applications of convolution

$A, B \subseteq \mathbb{Z},$ 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 $(v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)$

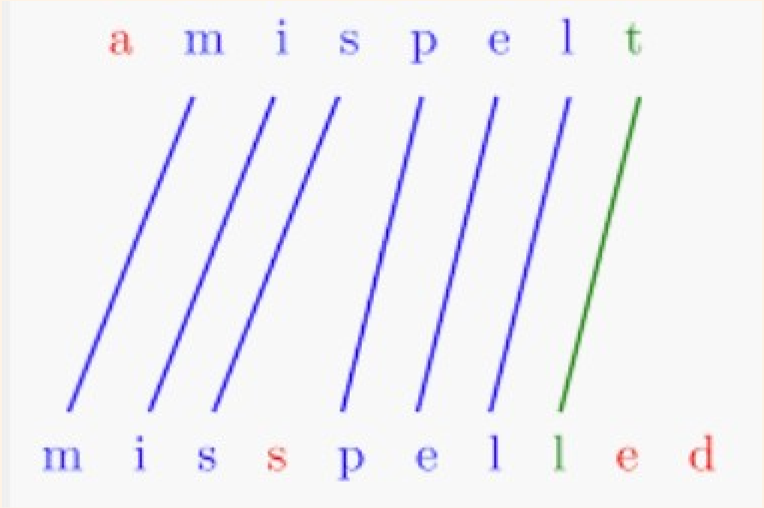
And a budget *W*, *among the points in*

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

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

$$\mathrm{supp}(u \star v) = \{i : (u \star v)_i \neq 0\}$$

$$(1 + x + x^{1001}) \cdot (-1 + x^4 + x^{10002})$$

Going beyond *O(d*log*d)*

-Boolean Convolution
-Nonnegative Convolution
-Most general case

An intermediate notion: cyclic convolution

$$u \star_c v \in \mathbb{R}^d$$

$$(u \star_c v)_i = \sum_{j=0}^{d-1} u_j v_{(i-j) \bmod d}$$

Hashing the support

Approach: Get our hands on the support of the convolution by performing a $\ll d$-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:

$$\widetilde{v}, \widetilde{u} \in \mathbb{R}^B$$

Short: sum up every B-th entry to obtain a B-length vector

$$\widetilde{v}_\ell = \sum_{i \in [d] : i \bmod B = \ell} v_i$$

$$\widetilde{u}_m = \sum_{i \in [d] : i \bmod B = \ell} u_i$$

Claim: $\widetilde{v} \star_c \widetilde{u}$ Resembles a *hashing* of $u \star v$ to B buckets.

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:

$$(\widetilde{u} \star_c \widetilde{v})_r = \sum_{i \in [d] : i \bmod \mathrm{B} = r} (u \star v)_i$$

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

$$(j, i-j) \mapsto (j \bmod \mathrm{B}, (i-j) \bmod B)$$

$$(j \bmod \mathrm{B} + (i-j) \bmod B) \bmod \mathrm{B} = i \bmod \mathrm{B}$$

It contributes to the ***(i mod B)***-th term in the **folded** version

Convolution preserving

Hashing the support

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

$$(\widetilde{u} \star_c \widetilde{v})_r = \sum_{i \in [d] : i \bmod \mathrm{B} = r} (u \star v)_i$$

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 ~ |T| $(\log d)^2$

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

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

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

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 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 $A+B= \{a+b \mid a \text{ in } A, b \text{ in } B\}$
in time $O(|A+B|(\log d)^5)$, for A,B *in* $[d]$.

(correctness) $A + B \subseteq T$

(time bound) $|T| \leq 3|A' + B'| \leq 3|A + B|$

$$A' = \left\{ \left\lfloor \frac{a}{2} \right\rfloor | a \in A \right\}$$

$$B' = \left\{ \left\lfloor \frac{b}{2} \right\rfloor | b \in B \right\}$$

$\longrightarrow$ Compute $A'+B'$ **recursively** $\longrightarrow$ $T := 2 \cdot (A' + B') + \{0, 1, 2\}$

$\downarrow$

Compute $A+B$ using the (intermediate) Promise problem rou

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
(size of input)+ (size of output)

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

$$\tilde{v}_m = \sum_{i \in [d] : i \bmod \mathrm{B} = m} v_i \cdot \omega^i$$

$$(\tilde{u} \star \tilde{v})_r = \sum_{i \in [d] : i \bmod B = r} (u \star v)_i \cdot \omega^i$$

$$\tilde{u}_\ell = \sum_{i \in [d] : i \bmod \mathrm{B} = \ell} u_i \cdot \omega^i$$

If *i* is isolated, then from the complex part of ,we can read ω^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 out = size of output, we'd set *B = 10out (log d)*^{2}

We recover a constant fraction of coordinates + introduce false positives
... so "recurse" to clean the erros.

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