COMS 4995-8: Advanced Algorithms (Spring'21) Feb 11, 2021

Lecture 10: Locality Sensitive Hashing, Graphs

Instructor: *Alex Andoni* Scribes: *Rishav (rk3142), Arjun (aa4664)*

1 Overview of Lecture

Today's lecture is a continuation of our discussion in nearest neighbour search. We will finish the topic, discussing about Local Sensitive Hashing for Hamming Space and then move to graphs.

2 c-ANN for Hamming Space

Family $\mathcal{H}$ of $h : \{0,1\}^d \to U$ is a $(r, cr, P_1, P_2) - LSH$ if: $\forall p, q \in \{0,1\}^d$:

$$\|p - q\|_1 \leq r \implies Pr_h[h(p) = h(q)] \geq P_1$$

$$\|p - q\|_1 > cr \implies Pr_h[h(p) = h(q)] < P_2$$

Fact 1. *If $P_2 < 1$, then $P_1 < 1$.*

The above fact is an example of an isoperimetric statement (and stronger, more quantative statements are known).

Theorem 2. *If there exists (r, cr, P_1, P_2)-LSH, then we can solve **C-ANN** (C-Approximate Nearest Neighbour) with the following parameters:*

1. ***Space**: $O(nd + n^{\frac{1+\rho}{P_1}})$*
2. ***Query Time**: $O(n^{\frac{\rho}{P_1}})$.d.[1 + time to compute h(q)]*

Where $\rho = \frac{log\frac{1}{P_1}}{log\frac{1}{P_2}} \in (0,1)$. $\rho \longrightarrow$ Ratio of the exponents of the probabilities.

Observation 3. *Let us fix $k \in \mathcal{N}$ and define a new hash function $g(p) = \langle h_1(p), h_2(p), \ldots, h_k(p) \rangle$ where $h_1, \ldots h_k \in \mathcal{H}$ iid. $g : \{0,1\}^d \to U^k$. Then g(p) is defined as follows:*

$$g(p) = h_1(p) \cdot h_2(p) \cdot \ldots, h_k(p) \in U^k; \text{ where . denotes concatenation}$$

The above is an LSH with parameters (r, cr, P_1^k, P_2^k). If p, q are close:

$$Pr[h_i(p) = h_i(q)] = P_1$$

$$Pr_g[g(p) = g(q)] = \prod_{i=1}^{k} Pr[h_i(p) = h_i(q)]$$

$$Pr_g[g(p) = g(q)] = P_1^k$$

Algorithm

1. **Preprocessing**: Build a dictionary data structure on points g(p), p ∈ D(Dataset)
2. **Query q**: Compute g(q)
   (a) Retrieve all points p ∈ D such that g(p) - g(q), using dictionary data structure
   (b) Enumerate through n until find a point p with $\|q - p\| \leq cr$.
3. **Performance Analysis**
   (a) **Space**: dictionary is stored on n points ⟶ O(n) [O(nd) to store original points]
   (b) **Query Time**: Time to compute g(q) + time taken to compute distance calculations in the bucket.

$\mathbb{E}$[# distance calculations] ≤ 1+E[# distance calculation to points $p \in D : \|p - q\| \geq cr$ and $h(p) = h(q)$]

$$\leq 1 + n \cdot P_r[g(p) = g(q)][||p - q|| \geq cr]$$

By LSH property

$$\leq 1 + n * P_2^k$$

Proof. Assuming there exists p^* at a distance ≤ r from q. We will find an ANN. To ensure this algorithm is correct, we need to show that p^* will be in the bucket. That is:

$$P_r[\text{algorithm outputs a cr-nearest-neighbor}] \geq P_r[p^*) = g(q)] \geq P_1^k$$

If we set k to be sufficiently large, then P_1^k will be small. The probability of correctness could be very small. To improve the probability of success, we will repeat the above process L times (each with a fresh function g) where L is:

$$L = \frac{10}{P_1^k}$$

Space : $O(Ln + nd)$
Query Time : $O(L \cdot (1 + n \cdot P_2^k + [\text{Time to compute g(q)}]))$

$$P_r[success] \geq [(1 - P_1^k)^L] \approx 1 - \exp^{\left(\frac{-10}{P_1^k} \cdot P_1^k\right)} \geq 0.9$$

The value of L depends purely on k. In order to minimize the Query time, we need to set k such that:

$$P_2^k = \frac{1}{n}$$

$$k = \lceil \frac{logn}{log\frac{1}{P_2}} \rceil$$

Remarks 1. *Note: k has to be an integer, so we add a ceil to the above equation.*

$$L = O\left(\frac{1}{P_1^k}\right) = O\left(\frac{1}{P_2^{\left(k \cdot \frac{\log\frac{1}{P_1}}{\log\frac{1}{P_2}}\right)}}\right) = O(n^\rho) \tag{1}$$

Observation 4. *The query time is always sub-linear in n as long as we can get $P_1 > P_2$.*

LSH family for $\{0,1\}^d$, IM' 98

We can construct an LSH for Hamming space $\{0,1\}^d$. Define $h(p) = P_j$ where j is random (*picked* before seeing the data). Hash function g is usually a concatenation of *primitive* functions. That is, $g(p) = \langle h_1(p), h_2(p), \ldots, h_k(p)\rangle$. Then, $g(p)$ is just a projection on k random bits and

$$Pr[h(p) = h(q)] = 1 - \frac{Ham(p,q)}{d}$$
$$P_1 = 1 - \frac{r}{d} \approx e^{-r/d}$$
$$P_2 = 1 - \frac{cr}{d} \approx e^{-cr/d}$$

Hence,

$$\rho = \frac{\log(1/P_1)}{\log(1/P_2)} = \frac{r/d}{cr/d} = \frac{1}{c}$$

Corollary 5. *C-ANN for a Hamming space $\{0,1\}^d$ has*
***Space**: O(nd + $n^{(1+\frac{1}{c})}$)*
***Query Time**: O($n^{\frac{1}{c}}\dot{d}$)*

Example 6. *For c = 2,*
***Space**: O($n^{1.5}$)*
***Query Time**: O($\sqrt{n}d$)*
Thus, the 2-approximation of ANN on Hamming Space can be solved in $\sqrt{n}$ time.

Remarks 2. *Can we do better? Is there any better way to partition the Hamming space so that we have better properties. No we can't do so, as mentioned in the below theorem.*

Theorem 7 (MNP '06, OWZ '10)**.** *The best possible cases:*

$$\textit{For the Hamming space}, \rho \geq \frac{1}{c}$$

$$\textit{For the Euclidean space } l_2, \rho \leq \frac{1}{c^2}$$

Remarks 3. *It is possible to beat these lower bounds by considering data-dependent LSH(≈ Perfect Hashing)*

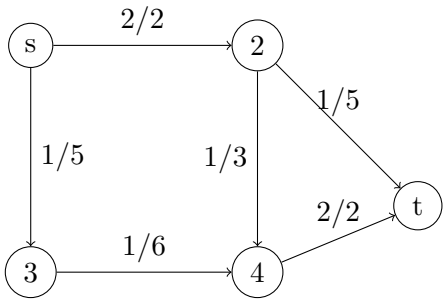
Remarks 4. *We can trade-off space vs query time to obtain:*
***Space**: $n^{(1+\rho_u)}$*
***Query Time**: n_q^{ρ}*

1. *It is possible to push the query time to be close to a constant, but then the space will increase.*
2. *It is possible to obtain a linear space. Then $\rho_u = 0$ and Query time will be sub-linear. Eg. $\rho_u = 0$, then $\rho_q < 1$.*

3 Graphs

In this section we will be seeing algorithms which are exact,i.e., neither random nor approximate. We will talk about the Max-Flow problem. It illustrates a number of techniques which are applicable for Graph Algorithms.

Consider, G:(V,E,C), where V is the number of vertices, E is the number of edges, C is Capacity. We will assume $\forall c_e \in C \geq 0$. Example:



The numerator in the weights denote the unit that flows from one node to another. Fix source(s) and destination(t). f is a flow vector $f \in \mathbb{R}_t^m$, m $\rightarrow$ number of edges, n $\rightarrow$ number of nodes, such that:

1. $f_e \geq 0, \forall e \in E$
2. $f_e \leq C_e, \forall e \in E$
3. Flow conservation; A flow that satisfies the below equation is called a valid flow.

$$\sum_{(u,v)\in E} f_{(u,v)} - \sum_{(v,u)\in E} f_{(v,u)} = 0 \ ; \ \forall v \neq s,t$$

Example 8. *f = all 0's is a valid flow.*

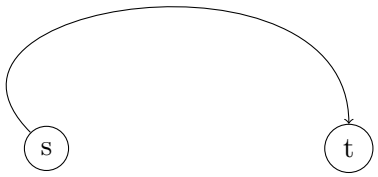
Max-Flow Problem: Find f maximizing flow shipped from s to t:

$$|f| = \sum_{(s,u)\in E} f_{(s,u)} - \sum_{(u,s)\in E} f_{(u,s)} = \sum_{(t,u)\in E} f_{(t,u)} - \sum_{(u,t)\in E} f_{(u,t)} \tag{2}$$

We can prove $|f| = \sum_{(t,u)\in E} f_{(t,u)} - \sum_{(u,t)\in E} f_{(u,t)}$ for any flow by summing up all flow contribution constraints for $v \neq s,t$; $\forall edge(u,v)$ not incident on s,t will cancel out.

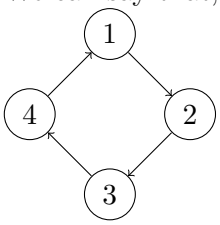
Other types of flow

Definition 9. *Path Flow: A particular type of flow which is a path. For example: There is a path from node s to node t in the figure below.*



Definition 10. *Cycle Flow: A particular type of flow that starts from a node and comes back into that node.*

We can say that, any flow that doesn't split or combine is either a path flow or a cycle flow.

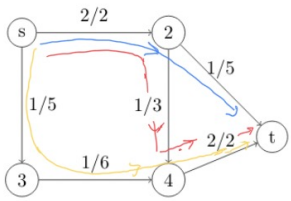


Theorem 11. *Any valid flow f can be decomposed into a set of path flows $f_{p_1}, f_{p_2} \dots f_{p_k}$ and cycle flows $f_{c_1}, f_{c_2} \dots f_{c_l}$ s.t.*

$$f = f_{p_1}, f_{p_2} \dots f_{p_k} + f_{c_1}, f_{c_2} \dots f_{c_l}; \ \ k + l \leq m$$

Example 12. *Refer to the following image which shows the path flows from s to t.*

In the **next class**, we shall prove this theorem and will study Ford-Fulkerson algorithm.


s
2
3
4
t
2/2
1/5
1/5
1/3
2/2
1/6