CS 38 Introduction to Algorithms **Week 3 Recitation Notes**

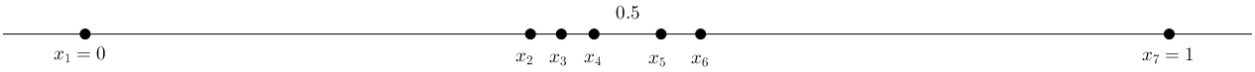
Greedy Algorithms

TA: Siddharth Prasad (sprasad@caltech.edu)

1 Warm-up

Problem Describe an efficient algorithm that, given a set $X = \{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points.

First (Incorrect) Attempt The topic here is greedy algorithms, so an attractive algorithm might be to simply choose intervals that maximize the number of points that can be covered. Why this algorithm fails is illustrated by the following picture:



Our idea will choose an interval to cover points $x_2, \ldots, x_6$, and will then require two extra intervals to cover x_1 and x_7, while the optimal solution uses only two intervals: one to cover points $x_1, \ldots, x_4$, and one to cover points $x_5, \ldots, x_7$.

Second Idea - Algorithm Sort and re-label the points so that $x_1 \leq x_2 \leq \cdots \leq x_n$. Let S be the set of intervals we want to build. For each $1 \leq j \leq n$, let $I_j = [x_j, x_j + 1]$. Our algorithm works as follows:

1. Initialize $S = \emptyset$.
2. For each $x_k \in X$ (while $X \neq \emptyset$):
   (a) $S \leftarrow S \cup \{I_k\}$
   (b) $X \leftarrow X \setminus \{x_\ell : x_\ell \in I_k\}$
3. Return S.

Proof. We will prove that our solution satisfies the so called "greedy choice property". We demonstrate that there is an optimal set of intervals that contains the interval $I_1 = [x_1, x_1 + 1]$. Suppose S_{opt} is an optimal set of intervals such that x_1 is covered by the interval $[p, p+1]$ where $p < x_1$. As x_1 is the leftmost point, there are no points from X contained in the interval $[p, x_1)$. Therefore, we could simply replace the interval $[p, p+1]$ in S_{opt} with the interval $[x_1, x_1 + 1]$ such that the new set of intervals S is still optimal (as $|S| = |S_{opt}|$ and all points are covered).

Then, an optimal solution to the problem can be built by solving the subproblem with all points in $[x_1, x_1 + 1]$ removed – say this yields an optimal solution S' – and considering $S' \cup \{[x_1, x_1 + 1]\}$. The greedy approach works for the subproblems by an identical argument applied to the points greater than $x_1 + 1$.

The time complexity of this algorithm is dominated by sorting the input, which can be done in $O(n \log n)$ time. □

2 Set Cover Approximation

Set Cover Problem We are given a pair $(X, \mathcal{F})$ where X is a finite set and $\mathcal{F}$ is a family of subsets of X such that for every $x \in X$ there is $F \in \mathcal{F}$ such that $x \in F$. So clearly $X = \cup_{S \in \mathcal{F}} S$.

The set cover problem asks us to find a subset $C \subseteq \mathcal{F}$ of minimum size such that $X = \cup_{S \in C} S$. You know from CS 21 that Set Cover is NP-hard. We cannot expect to write an efficient algorithm to solve this problem, so we present an approximate one.

Greedy Algorithm The rough idea is we greedily construct our set cover by choosing the subset of $\mathcal{F}$ that covers the largest number of remaining uncovered elements.

1. Initialize $C = \emptyset$.
2. While $X \neq \emptyset$:
   (a) Choose $S \in \mathcal{F}$ such that $|S \cap U|$ is maximized.
   (b) $X \leftarrow X \setminus S$
   (c) $C \leftarrow C \cup \{S\}$
3. Return C.

This algorithm has a polynomial runtime in $|X|$ and $|\mathcal{F}|$. It is straightforward to see that this algorithm successfully returns a set cover of X, but it is clearly not optimal. However we can prove that the greedy algorithm is always very close to being optimal.

Approximate Solution Suppose an optimal set cover is C^* and our algorithm outputs a set cover $C \geq C^*$. While our algorithm is not always optimal, we will show that the ratio $|C|/|C^*|$ is never too large, so the algorithm is only some small factor away from an optimal solution.

Suppose S_i is the ith subset chosen by our algorithm, and let us charge the algorithm a unit cost every time it adds a set to C. We then distribute this cost evenly to all the elements covered by the set chosen that have NOT been covered before. So if $x \in X$ is covered by S_i for the first time, we have $cost(x) = \frac{1}{|S_i \setminus (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$.

Then, the total cost incurred by the algorithm is $\sum_{x \in X} cost(x)$, but since the algorithm incurs one unit of cost every time it adds a set to the set cover, this is also $|C|$. So we have $|C| = \sum_{x \in X} cost(x)$. Combining this with the obvious inequality $\sum_{s \in C^*} \sum_{x \in S} cost(x) \geq \sum_{x \in X} cost(x)$, we get the inequality

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} cost(x). \tag{1}$$

We will now attempt to bound the quantity $\sum_{x \in S} cost(x)$ for any $S \in \mathcal{F}$. Fix an arbitrary $S \in \mathcal{F}$. Let u_i be the number of elements in S that have not been covered yet after i iterations of the algorithm. So $u_i = |S \setminus (S_1 \cup S_2 \cup \cdots \cup S_i)|$ and $u_0 = |S|$. Clearly $u_i \leq u_{i-1}$ and when our algorithm chooses set S_i, S_i covers $u_{i-1} - u_i$ elements that have not been previously covered.

Now, if m denotes the smallest index such that $u_m = 0$, then

$$\sum_{x \in S} cost(x) = \sum_{i=1}^{m} (u_{i-1} - u_i) \cdot \frac{1}{|S_i \setminus (S_1 \cup \cdots \cup S_{i-1})|} \leq \sum_{i=1}^{m} (u_{i-1} - u_i) \cdot \frac{1}{|S \setminus (S_1 \cup \cdots \cup S_{i-1})|} = \sum_{i=1}^{m} (u_{i-1} - u_i) \cdot \frac{1}{u_i}.$$

It is a straightforward calculation to show that this sum is at most $H(u_0)$, where $H(d) = 1 + 1/2 + \cdots + 1/d$ is the dth harmonic number.

Then, going back to our bound (1), we have

$$|C| \leq \sum_{S \in C^*} \sum_{x \in S} cost(x) \leq \sum_{s \in C^*} H(|S|) \leq |C^*| \cdot H(\max\{|S| : S \in F\}).$$

Thus, we have shown that the greedy algorithm for Set Cover is a polynomial time $(\log |X|)$-approximation algorithm.

3 Matroids

Definition 3.1. *A* matroid *is an ordered pair $M = (S, \mathcal{I})$ where S is a finite set and $\mathcal{I}$ is a nonempty family of subsets of S (called the independent subsets of S) satisfying the following properties:*

1. *Hereditary property: If $B \in \mathcal{I}$ and $A \subseteq B$, then $A \in \mathcal{I}$. Note that this implies $\emptyset \in \mathcal{I}$.*
2. *Exchange property: Suppose $A, B \in \mathcal{I}$ with $|A| < |B|$. Then, there is an $x \in B \backslash A$ such that $A \cup \{x\} \in \mathcal{I}$.*

Basis for a matroid A *basis* for a matroid $(S, \mathcal{I})$ is a maximal independent set, i.e. an independent set $B \in \mathcal{I}$ such that for all points $x \notin B$, we have $B \cup \{x\} \notin \mathcal{I}$.

Theorem 3.2. *If B_1 and B_2 are bases for a matroid $(S, \mathcal{I})$, then $|B_1| = |B_2|$.*

Proof. Suppose for the sake of contradiction that (WLOG) $|B_1| > |B_2|$. By the exchange property, there is some $x \in B_1 \setminus B_2$ such that $B_2 \cup \{x\}$ is independent, which contradicts that B_2 is a basis. □

Example 2. *Let S be a finite set of vectors, and let the family of independent sets $\mathcal{I}$ consist of all linearly independent subsets of vectors from S. It is clear that a basis for the matroid is exactly a basis in the usual sense for* span(S).

Weighted matroids A matroid $M = (S, \mathcal{I})$ is *weighted* if we can associate with the matroid a weight function $w : S \to \mathbb{R}^+$ that assigns a positive weight to every element of S. The weight function can be easily extended to independent sets by $w(A) = \sum_{x \in A} w(x)$.

Max-weight Matroid Basis The problem of choosing a basis B of a weighted matroid $M = (S, \mathcal{I})$ with weight function w such that the weight $w(B) = \sum_{x \in B} w(x)$ is maximized lends itself nicely to a greedy analysis.

Greedy Algorithm The following easy to state greedy algorithm solves the max-weight basis choice problem for matroids. For a proof see Lemma 16.7 in CLRS.

1. Initialize $B = \emptyset$, and sort S in decreasing order by weight.
2. For each $x \in S$:
   (a) If $B \cup \{x\} \in \mathcal{I}$ then $B \leftarrow B \cup \{x\}$.
3. Return B.

It is straightforward to see that the running time of this routine is $O(n \log n + nf(n))$, where $f(n)$ is the time required to check whether $B \cup \{x\}$ is independent.

Graphic Matroid Graphic matroid $M_G = (S_G, \mathcal{I}_G)$ defined for an undirected graph $G = (V, E)$ as follows: $S_G = E$. The independent sets are subsets $A \subset E$ such that A contains no cycles. In other words A is independent iff the induced subgraph $G_A = (V, A)$ is a forest. Proof that M_G is indeed a matroid is Theorem 16.5 in CLRS. Note that if M_G is the graphic matroid for a connected undirected graph G, the bases for M_G are exactly the spanning trees of G.

MST as a Max-weight basis problem It is suggestive from the previous discussion that we can cast the problem of finding an MST in a connected undirected graph as a max-weight basis choice problem in a suitably defined weighted matroid. We are given an undirected graph G and a weight function $w : E \to \mathbb{R}^+$. Consider the graphic matroid M_G with weight function w' defined by $w'(e) = w_0 - w(e)$, where $w_0 > \max_{e \in E} w(e)$. Let B be a basis of M_G. We will show that finding a max-weight basis for M_G is precisely the problem of finding a MST for G. We have

$$w'(B) = \sum_{e \in B} w'(e) = \sum_{e \in B} (w_0 - w(e)) = (|V| - 1)w_0 - \sum_{e \in B} w(b) = (|V| - 1)w_0 - w(B).$$

It is clear now that maximizing $w'(B)$ is equivalent to minimzing $w(B)$.