Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute of Theoretical Computer Science
Bernd Gärtner, Mohsen Ghaffari, Rasmus Kyng, David Steurer

Algorithms, Probability, and Computing Solutions KW43 HS20

Solution 1

This is the classical use case for backward analysis. Instead of considering the points as being inserted one by one and counting the number of nearest neighbors occurring, we get the very same number if we start with the whole set $P_n := P$, and then for each $i = n-1, n-2, \dots, 1$, obtain P_i from P_{i+1} by removing a point $p \in P_i$ chosen uniformly at random.

Define X_i to be the nearest neighbor of q in P_i. We are interested in the random variable $X := |\{X_i | i \in \{1..n\}\}|$. To this end we just calculate the probability $x_i = \Pr[X_i \neq X_{i+1}]$ for $1 \leq i \leq n-1$. Clearly, then,

$$\mathbf{E}[X] = 1 + \sum_{i=1}^{n-1} x_i.$$

We now note that $X_i \neq X_{i+1}$ occurs if and only if $P_i = P_{i+1} \setminus \{X_{i+1}\}$. This yields that

$$x_i = \Pr[X_i \neq X_{i+1}] = \Pr[P_i = P_{i+1} \setminus \{X_{i+1}\}] = \frac{1}{i+1},$$

where the second equality follows from the fact that P_i is generated from P_{i+1} by removing a point uniformly at random. As a very important remark: please note that this simple argument only works because **no matter** what P_{i+1} is, P_i is always generated by removing one of its elements uniformly at random and the random variable X_i **is completely determined** by P_i (what the currently nearest point is does not depend on the insertion history). You have to verify this before you apply a backward analysis of this type.

Finally,

$$\mathbf{E}[X] = 1 + \sum_{i=1}^{n-1} x_i = H_n.$$

Solution 2

Given a linear program in standard form[1],

$$\text{maximize } c^T x \text{ subject to } Ax \leq b, \tag{LP 1}$$

[1] You might want to revise at this point how to convert any linear program into standard form (section 6.1 of the lecture notes).

where $A \in \mathbf{R}^{m\times n}$, $c \in \mathbf{R}^n$ and $b \in \mathbf{R}^m$, we can convert it into equational form as follows: First we replace the "$\leq$" by a "$=$" by the following trick.

$$\text{maximize } c^T x \text{ subject to } Ax + \varepsilon = b,\ \varepsilon \geq 0 \tag{LP 2}$$

where $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_m)$ is a vector of m new variables. In a second step we get rid of unconstrained (= possibly negative) variables. To this end we replace each x_i by $x_i' - x_i''$, where x_i', x_i'' are two new nonnegative variables:

$$\text{maximize } c^T(x' - x'') \text{ subject to } A(x' - x'') + \varepsilon = b,\ x' \geq 0,\ x'' \geq 0,\ \varepsilon \geq 0. \tag{LP 3}$$

Now we write this LP in such a way that it is undoubtedly in equational form, e. g. like this:

$$\text{maximize } \begin{pmatrix} c \\ -c \\ 0 \end{pmatrix}^T \begin{pmatrix} x' \\ x'' \\ \varepsilon \end{pmatrix} \text{ subject to } \begin{pmatrix} A & -A & \mathbf{I}_m \end{pmatrix} \begin{pmatrix} x' \\ x'' \\ \varepsilon \end{pmatrix} = b, \begin{pmatrix} x' \\ x'' \\ \varepsilon \end{pmatrix} \geq 0. \tag{LP 4}$$

In what sense have we "converted" the LP into equational form? We make the following notes.

- If x is a feasible solution of (LP 1), then a corresponding feasible solution of (LP 4) is given by

$$\begin{aligned} x' &= (\max\{x_1, 0\}, \ldots, \max\{x_n, 0\}), \\ x'' &= -(\min\{x_1, 0\}, \ldots, \min\{x_n, 0\}), \\ \varepsilon &= b - Ax. \end{aligned}$$

  Furthermore this feasible solution to (LP 4) has the same objective value as x.

- Correspondingly, if (x', x'', ε) is a feasible solution of (LP 4), then $x' - x''$ is a feasible solution of (LP 1) with the same objective value.

Complexity: The linear program (LP 4) has $2n + m$ variables and $2n + 2m$ constraints, where the original (LP 1) had n variables and m constraints.

Solution 3

(a) Suppose $R(\mathcal{S}) \neq \emptyset$. Clearly, this implies that $R(\mathcal{S} \setminus \{H\}) \neq \emptyset$. Therefore and since $H_1, H_2 \in \mathcal{S} \setminus \{H\}$, both $x := c(\mathcal{S})$ and $x' := c(\mathcal{S} \setminus \{H\})$ are well-defined points. Now there are two cases. Either $x' \in H$ or $x' \notin H$.

Suppose $x' \in H$. In that case, we readily have $x' \in R(\mathcal{S})$. And since x' was the point of smallest x-coordinate in $R(\mathcal{S} \setminus \{H\}) \supseteq R(\mathcal{S})$, it will a fortiori be the point of smallest x-coordinate in $R(\mathcal{S})$. Therefore $x' = x$ in that case.

Now suppose $x' \notin H$. In that case, we have $x' \notin R(\mathcal{S})$. Now consider the line segment s connecting x' and x. Since $x \in H$ but $x' \notin H$, s must cross ℓ_H at an intersection point p. Moreover, since $x', x \in R(\mathcal{S} \setminus \{H\})$ and since $R(\mathcal{S} \setminus \{H\})$ is a convex set, the whole line segment s runs inside $R(\mathcal{S} \setminus \{H\})$. Since $p \in H$ by construction, it follows $p \in R(\mathcal{S})$. Clearly, x' must have a strictly smaller x-coordinate than x, otherwise x would have been a possible choice for $c(\mathcal{S} \setminus \{H\})$ in the first place. Therefore, going further away from x' along s than p would only make x-coordinates larger, from which we conclude $p = x$ and thus $x \in \ell_H$.

(b) For any halfplane $G \in \mathcal{S} \setminus \{H\}$, let us say that G *points right (with respect to H)* if the intersection of G and ℓ_H is a ray that extends infinitely to the right. Similarly, let us say that G *points left (with respect to H)* if the intersection of G and ℓ_H is a ray that extends infinitely to the left. There is no third option because we assumed not to have vertical or parallel lines.

We can now proceed in the following way:

```
SolveInc. Input: $\mathcal{S}, H, x' = c(\mathcal{S} \setminus \{H\})$.
  (1) Check whether $x' \in H$. If so, return $x'$.
  (2) Else, initialize the (point) variables $h_l := (-\infty, 0)$ and $h_r := (+\infty, 0)$.
  (3) Now for all $G \in \mathcal{S} \setminus \{H\}$, do:
      (3.1) Let $y$ be the intersection point of the lines $\ell_G$ and $\ell_H$
      (3.2) If $G$ points right and $y$ has larger $x$-coordinate than $h_l$, update $h_l := y$
      (3.3) If $G$ points left and $y$ has smaller $x$-coordinate than $h_r$, update $h_r := y$
  (4) If $h_l$ has strictly larger $x$-coordinate than $h_r$, return 'empty', else return $h_l$.
```

That the running time is linear is trivial.

Now we argue for correctness. Suppose $R(\mathcal{S}) \neq \emptyset$. Since the conditions of (a) are then satisfied, we know that $c(\mathcal{S})$ is either equal to x' or on ℓ_H. In the first case, the algorithm returns on line (1). In the second case, the algorithm maintains two boundary points h_l and h_r such that on ℓ_H, exactly the points between h_l and h_r are admissible. Since we know $c(\mathcal{S}) \in \ell_H$ and $c(\mathcal{S}) \in R(\mathcal{S})$, we must have $c(\mathcal{S}) = h_l$ because this is the point of smallest x-coordinate within that set.

On the other hand if $R(\mathcal{S}) = \emptyset$, then we *must* end up in the situation that h_l lies right of h_r as otherwise h_l, for instance, would be contained in all halfspaces contrary to the assumption.

(c) Consider the following algorithm.

```
Solve. Input: $\mathcal{S}$
  (1) If $\mathcal{S} = \{H_1, H_2\}$, return $g$ as the certificate.
  (2) Else, select $H \in \mathcal{S} \setminus \{H_1, H_2\}$ uniformly at random.
  (3) Recursively call Solve$(S \setminus \{H\})$.
  (4) If the result is 'empty', return 'empty'
```

(5) Else, if the result is a point x', call '**SolveInc**$(\mathcal{S}, H, x')$ and return the result.

The base case is clearly fine. General correctness then follows by induction from the correctness of **SolveInc**.

The more difficult part will be to determine the running time. At first sight, it looks like this algorithm might require quadratic time; after all, we are calling **SolveInc** linearly many times and we know that **SolveInc** uses up to linear time in the worst case. The crucial observation is that **SolveInc**, on input $(\mathcal{S}, H, c(\mathcal{S}\setminus\{H\}))$, requires linear time only if $c(\mathcal{S}) \neq c(\mathcal{S}\setminus\{H\})$. If on the other hand those certificates are equal, then **SolveInc** can return on line (1) and takes only constant time to terminate.

Let us first only consider sets $\mathcal{S}$ such that $R(\mathcal{S}) \neq \emptyset$. Let t_n be expected running time for $n = |\mathcal{S}|$ such halfplanes. We have $t_2 = \mathcal{O}(1)$. Now for $n \geq 3$, t_n is bounded according to the above observation as

$$t_n \leq t_{n-1} + \Pr\left[c(\mathcal{S}) \neq c(\mathcal{S}\setminus\{H\})\right] \cdot \mathcal{O}(n) + \Pr\left[c(\mathcal{S}) = c(\mathcal{S}\setminus\{H\})\right] \cdot \mathcal{O}(1).$$

But what is the probability $\Pr\left[c(\mathcal{S}) \neq c(\mathcal{S}\setminus\{H\})\right]$? According to (a), this can happen only if $\ell_H \ni c(\mathcal{S})$. But since H is chosen uniformly at random from all halfplanes (except for H_1, H_2) and since $c(\mathcal{S})$ cannot lie on any more than two boundary lines, the probability that this happens is bounded by $\frac{2}{n-2}$. We therefore obtain that

$$t_n \leq t_{n-1} + \frac{2}{n-2} \cdot \mathcal{O}(n) + \mathcal{O}(1) = t_{n-1} + \mathcal{O}(1).$$

From this it readily follows that $t_n = \mathcal{O}(n)$ as desired.

The only thing we have left to do is to consider the case when the result will be '`empty`'. However, consider that as soon as a recursive step returns '`empty`', all higher recursion levels return in constant time. So the transition from a certificate point to the result '`empty`' happens at most once in the course of the algorithm. Before that, the bound we just derived applies. Therefore the total time is no bigger than linear in that case either.