CS303- Computational Geometry

Name: Niladri Ghosh
ID: B2430100

Date: 10 November, 2025
Assignment: Number 3

Problem-1

Ex.4.10 Let H be a set of at least three half-planes with a non-empty intersection such that not all bounding lines are parallel. We call a half-plane $h \in H$ *redundant* if it does not contribute an edge to H. Prove that for any redundant half-plane $h \in H$ there are two half-planes $h', h'' \in H$ such that $h' \cap h'' \subset h$. Give an $O(n \log n)$-time algorithm to compute all redundant half-planes.

Solution

We address the problem in two parts: first, by proving the geometric containment property for redundant half-planes, and second, by providing an efficient algorithm to identify them.

Proof of the Claim.

Let $P = \bigcap_{g \in H} g$ denote the feasible region formed by the intersection of the set H. Since the intersection is non-empty and not all bounding lines are parallel, P is a convex polygonal region (possibly unbounded). By definition, a half-plane $h \in H$ is *redundant* if $P = \bigcap_{g \in H \setminus \{h\}} g$. This implies that $P \subseteq h$, and the boundary line ∂h does not support an edge of P.

Consider the outward normal vector $\vec{n}$ of the redundant half-plane h. We define a linear function $f(x) = \vec{n} \cdot x$. Since P is defined by a finite set of linear inequalities, the maximum of $f(x)$ over the region P is attained at a vertex (or potentially an edge, but we may choose a vertex endpoint) of P. Let v be a vertex of P that maximizes $f(x)$.

Since v is a vertex of the polygonal region P, it is formed by the intersection of the boundaries of two distinct half-planes $h', h'' \in H$ that are essential (i.e., they contribute edges adjacent to v). Thus, $\{v\} = \partial h' \cap \partial h''$. The intersection $h' \cap h''$ forms a wedge with apex v containing P. Since h contains P and its boundary ∂h lies "beyond" v in the direction of $\vec{n}$ (or passes through v without cutting into the interior of the wedge defined by P), the constraints imposed by h' and h'' are strictly tighter than or equivalent to h in the vicinity of v. Consequently, the entire wedge formed by h' and h'' must lie inside h. Therefore:

$$h' \cap h'' \subset h.$$

Algorithm.

To compute all redundant half-planes in $O(n \log n)$ time, we effectively compute the half-plane intersection P and report all half-planes that do not contribute an edge to its boundary. We utilize a **Sort-and-Scan** approach similar to the Graham scan for convex hulls:

1. **Filter Parallel Lines:** If multiple half-planes have normal vectors with the same polar angle, we retain only the one with the most restrictive constraint (i.e., the line furthest "inward"). All others are immediately marked as redundant. This takes $O(n \log n)$ or $O(n)$ depending on the sorting method.

2. **Sort:** Sort the remaining unique half-planes by the polar angle of their normal vectors in the range $(-\pi, \pi]$. This step requires $O(n \log n)$ time.

3. **Deque Scan:** We maintain a double-ended queue (deque) of half-planes representing the current intersection chain. We iterate through the sorted half-planes. For each new half-plane h_{new}:
   - While the intersection point of the last two half-planes in the deque lies strictly outside h_{new}, remove the last half-plane from the deque (it has become redundant).
   - While the intersection point of the first two half-planes in the deque lies strictly outside h_{new}, remove the first half-plane.
   - Add h_{new} to the back of the deque.

4. **Cleanup and Output:** After processing all planes, we perform a final check on the deque boundaries to ensure the chain closes correctly (checking the first intersection against the last plane and vice versa). The half-planes remaining in the deque are the essential ones.

The set of redundant half-planes is the complement of the set in the deque. Since the sorting step dominates the complexity and the scan is linear (each plane is pushed and popped at most once), the total time complexity is $O(n \log n)$.

Problem 2

Ex.4.14 Here is a paranoid algorithm to compute the maximum of a set A of n real numbers:

Algorithm ParanoidMaximum(A)

1: **if** card(A) = 1 **then**
2: **return** the unique element $x \in A$
3: **else**
4: Pick a random element x from A.
5: $x' \leftarrow$ ParanoidMaximum($A \setminus \{x\}$)
6: **if** $x \leq x'$ **then**
7: **return** x'
8: **else**
9: Now we suspect that x is the maximum, but to be absolutely sure, we compare x with all card(A) $- 1$ other elements of A.
10: **return** x

What is the worst-case running time of this algorithm? What is the expected running time (with respect to the random choice in line 3)?

Solution

We analyze the running time $T(n)$ based on the number of comparisons performed.

Worst-Case Running Time.

The worst case occurs when the algorithm is forced to perform the "paranoid" verification step (lines 7–8) as often as possible. This step costs $n - 1$ comparisons. The algorithm enters this expensive block only if the condition $x \leq x'$ is false, which implies $x > x'$. Since x' is the maximum of the remaining set $A \setminus \{x\}$, $x > x'$ implies that our randomly chosen x happens to be the true maximum of the entire set A.

If we are extremely unlucky and consistently pick the maximum element at every level of recursion, we pay the cost of $n-1$ comparisons at level n, then $n-2$ at the next level, and so on. The recurrence becomes:

$$T(n) = T(n-1) + O(n)$$

Summing this arithmetic series gives:

$$T(n) = \sum_{i=1}^{n}(i-1) \approx \frac{n^2}{2} = O(n^2)$$

Thus, the worst-case running time is quadratic.

Expected Running Time.

In the average case, we consider the probability of entering the expensive branch. At any step with n elements, we pick x uniformly at random. The expensive verification only happens if x is the maximum of A. The probability of picking the maximum element from n elements is $1/n$.

The recurrence for the expected running time $E[T(n)]$ involves the cost of the recursive call, one mandatory comparison (line 5), and the expected cost of the paranoid check:

$$E[T(n)] = E[T(n-1)] + \underbrace{1}_{\text{check } x \leq x'} + \underbrace{\frac{1}{n}(n-1)}_{\text{paranoid check}}$$

The term $\frac{1}{n}(n-1)$ is the probability of being paranoid $(1/n)$ multiplied by the cost of the check $(n-1)$. Simplifying the recurrence:

$$E[T(n)] = E[T(n-1)] + 1 + 1 - \frac{1}{n} < E[T(n-1)] + 2$$

Since the work added at each step is bounded by a constant (roughly 2 comparisons), the total expected running time is linear:

$$E[T(n)] = O(n)$$