CSE520: Computational Geometry
Lecture 13
Seidel's Linear Programming Algorithm

Antoine Vigneron

Ulsan National Institute of Science and Technology

June 15, 2020

Introduction

- References for this lecture: Textbook Chapter 4, Dave Mount's lecture notes lectures 8–10.

Algorithms for Linear Programming

- A practical algorithm: The simplex algorithm.
- Exponential time in the worst case.

- There are polynomial time algorithms: Interior point methods.
- For n constraints in dimension d, run in roughly $O(n^4 d)$ time.

- In this Lecture: Seidel's algorithm.
- A simple *randomized* algorithm.
- *Linear* time in low dimension.
- More precisely, $O(d!n)$. This is $O(n)$ when $d = O(1)$

- There is also a linear-time *deterministic* algorithm in low dimension:
- Megiddo's algorithm, which runs in $O(3^{d^2} n)$ time.
- Not covered in this course.

One dimensional case

Maximize

$$f(x) = cx$$

subject to

$$\begin{aligned} a_1 x &\leqslant b_1 \\ a_2 x &\leqslant b_2 \\ \vdots \quad &\vdots \\ a_n x &\leqslant b_n \end{aligned}$$

Interpretation

- If $a_i > 0$ then constraint i corresponds to the interval

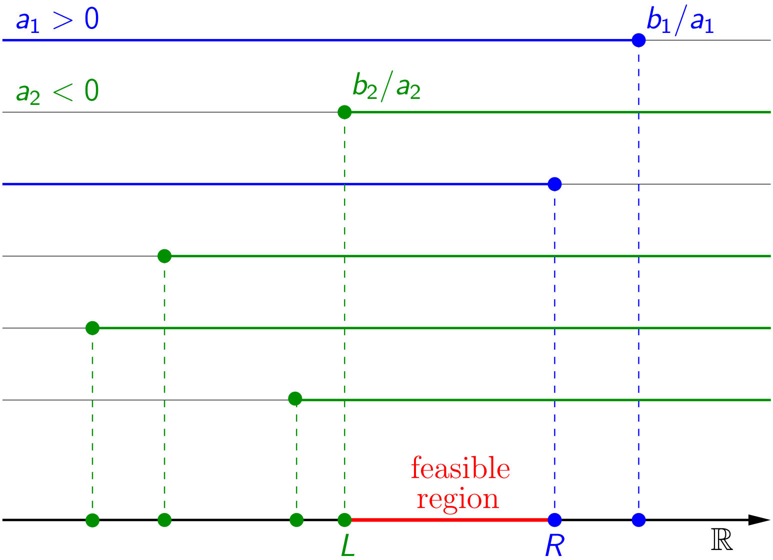
$$\left(-\infty, \frac{b_i}{a_i}\right].$$

- If $a_i < 0$ then constraint i corresponds to the interval

$$\left[\frac{b_i}{a_i}, \infty\right).$$

- The feasible region is an intersection of intervals.
- So the feasible region is an interval.

Interpretation



Algorithm

Case 1: $\exists(i_1, i_2)$ such that $a_{i_1} < 0 < a_{i_2}$.

- Compute
$$R = \min_{a_i > 0} \frac{b_i}{a_i}.$$
- Compute
$$L = \max_{a_i < 0} \frac{b_i}{a_i}.$$

- It takes $O(n)$ time. (No need to sort!)
- If $L > R$ then the program is infeasible.
- Otherwise
  - If $c > 0$ then the solution is $x = R$.
  - If $c < 0$ then the solution is $x = L$.

Algorithm

Case 2: $a_i > 0$ for all i.

- Compute $R = \min \frac{b_i}{a_i}$.
- If $c > 0$ then the solution is $x = R$.
- If $c < 0$ then the program is unbounded and the ray $(-\infty, R]$ is a solution.

Case 3: $a_i < 0$ for all i.

- Compute $L = \max \frac{b_i}{a_i}$.
- If $c < 0$ then the solution is $x = L$.
- If $c > 0$ then the program is unbounded and the ray $[L, \infty)$ is a solution.

Two-Dimensional Linear Programming

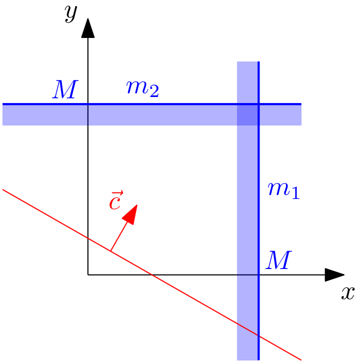
- First approach:
- Compute the feasible region.
  - The feasible region is a convex polygon (possibly unbounded).
  - Compute it in $O(n \log n)$ time by divide and conquer+plane sweep.
  - Other method: see later, lecture on duality.
- Find an optimal point.
  - Can be done in $O(\log n)$ time by binary search.
- Overall, it is $O(n \log n)$ time.

- This lecture: An expected $O(n)$-time algorithm.

Preliminary

- We only consider bounded linear programs.
- We can make sure that our linear program is bounded by enforcing two additional constraints m_1 and m_2.
  - Objective function: $f(x, y) = c_1 x + c_2 y$
  - Let M be a large number.
  - If $c_1 \geqslant 0$ then m_1 is $x \leqslant M$.
  - If $c_1 \leqslant 0$ then m_1 is $x \geqslant -M$.
  - If $c_2 \geqslant 0$ then m_2 is $y \leqslant M$.
  - If $c_2 \leqslant 0$ then m_2 is $y \geqslant -M$.

New constraints



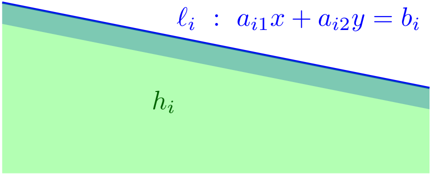
- In practice, it is often easy to set the constraints m_1 and m_2.
- For instance, in the first example, of the previous lecture, we can choose $M = 30$.

Notation

- The ith constraint is:

$$a_{i1}x + a_{i2}y \leqslant b_i.$$

- It defines a half-plane h_i.

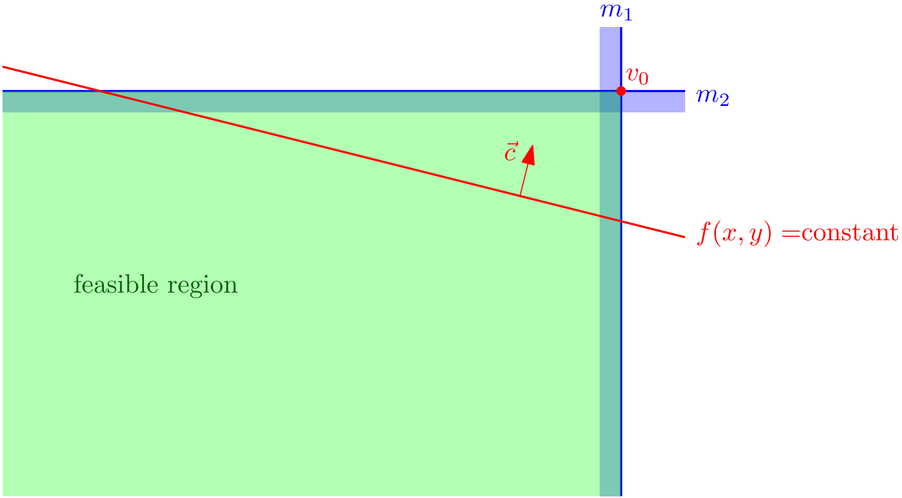


- Let ℓ_i denote the line that bounds h_i.

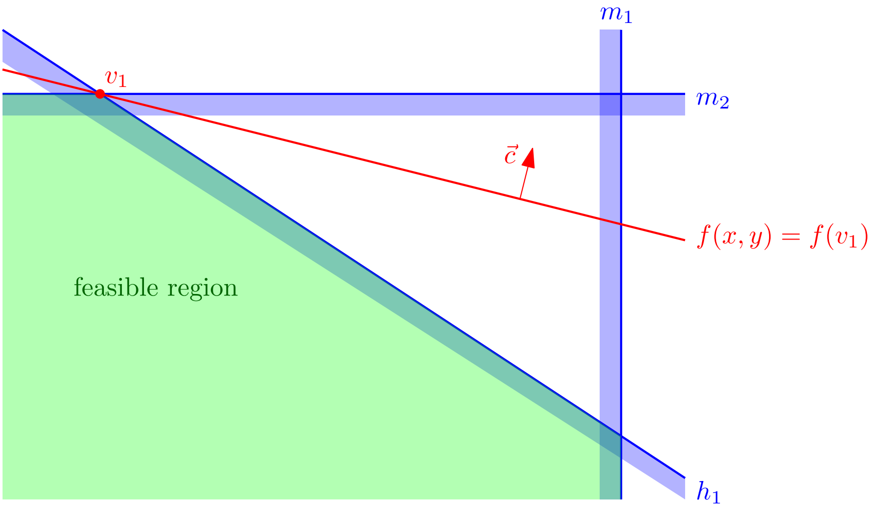
Algorithm

- A randomized incremental algorithm.
- We first compute a random permutation of the constraints $(h_1, h_2, \dots, h_n)$.
- We denote $H_i = \{m_1, m_2, h_1, h_2, \dots, h_i\}$.
- We denote by v_i a vertex of $\bigcap H_i$ that maximizes the objective function.
  - In other words, v_i is a solution to the linear program where we only consider the first i constraints.
  - v_0 is simply the vertex of the boundary of $m_1 \cap m_2$.
- Idea: knowing v_{i-1}, we insert h_i and find v_i.

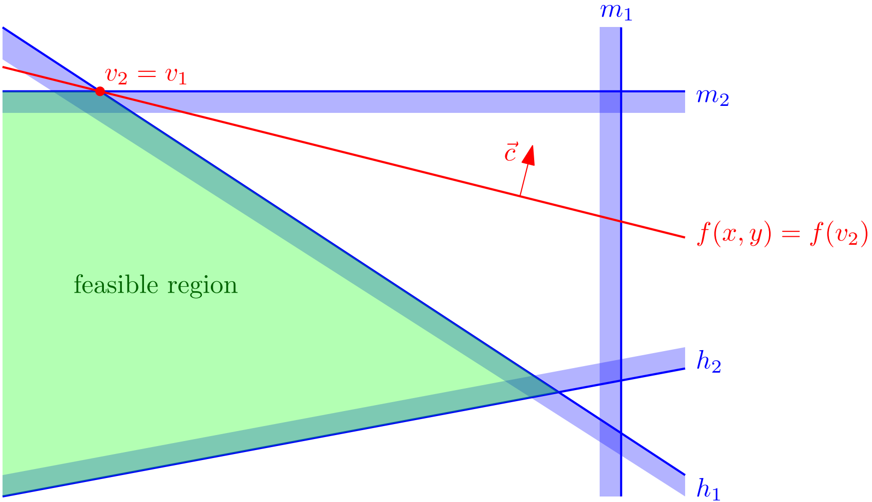
Example



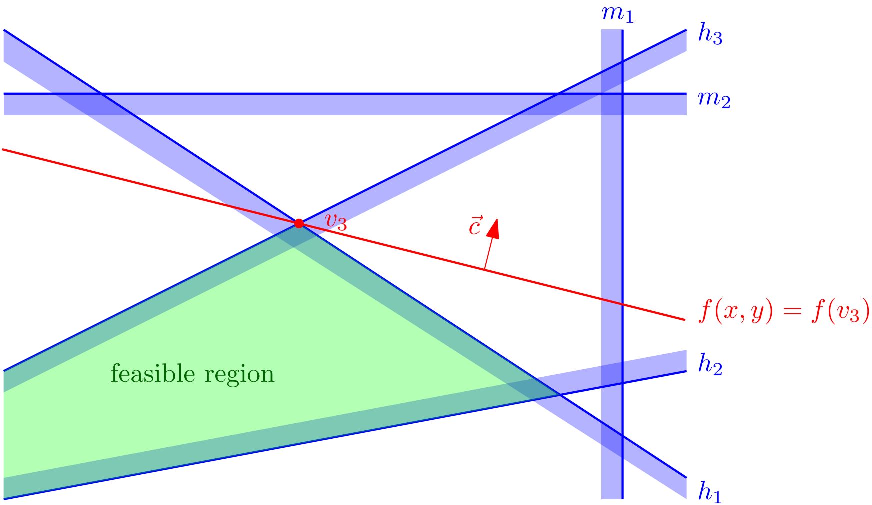
Example



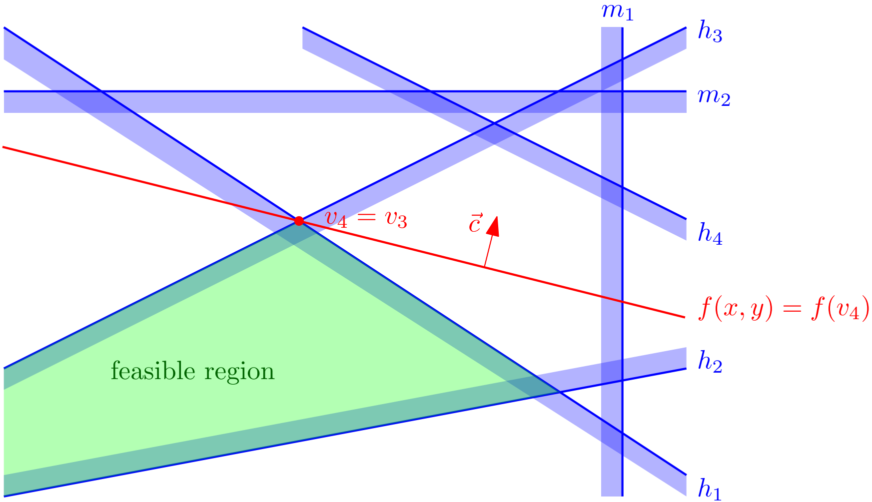
Example



Example



Example

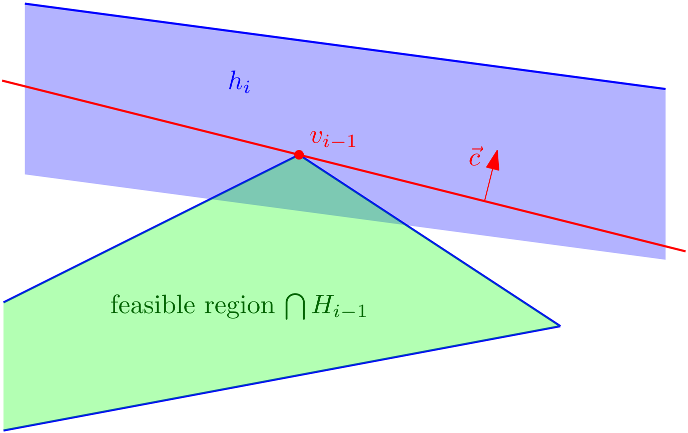


Algorithm

- Randomized incremental algorithm.
- Before inserting h_i, we only assume that we know v_{i-1}.
- How can we find v_i?

First Case

- First case: $v_{i-1} \in h_i$.

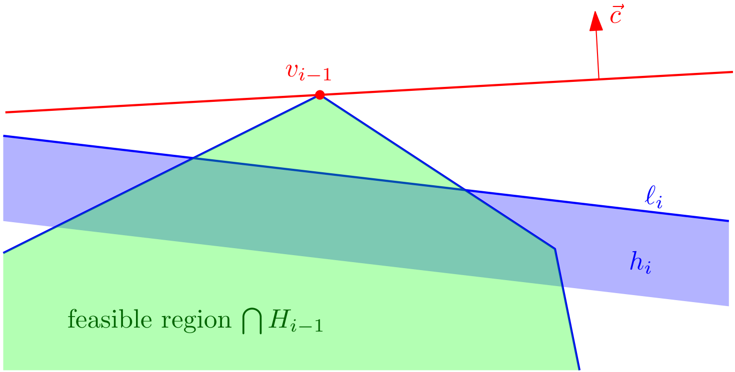


- Then $v_i = v_{i-1}$.

(Proof done in class.)

Second Case

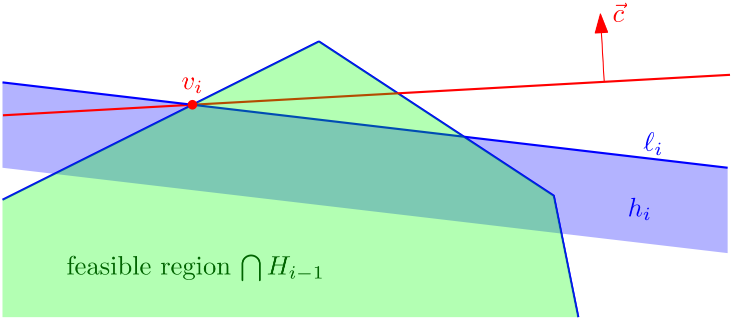
- Second case: $v_{i-1} \notin h_i$.



- Then v_{i-1} is not in the feasible region of H_i.
- So $v_i \neq v_{i-1}$.
- What do we know about v_i?

Second Case

- There exists an optimal solution $v_i \in \ell_i$.



- Proof?
- How can we find v_i?

Second Case

- Assume that $a_{i2} \neq 0$, then the equation of ℓ_i is
$$y = \frac{b_i - a_{i1}x}{a_{i2}}.$$
- We replace y with $\frac{b_i - a_{i1}x}{a_{i2}}$ in all the constraints of H_i and in the objective function.
- We obtain a one dimensional linear program, with variable x.
- If it is feasible, its solution gives us the x-coordinate of v_i.
- We obtain the y-coordinate using the equation of ℓ_i.
- If this linear program is infeasible, then the original 2D linear program is infeasible too and we are done.

Analysis

- First case is done in $O(1)$ time: Just check whether $v_{i-1} \in h_i$.
- Second case in $O(i)$ time: One dimensional linear program with $i+1$ constraints.
- So the algorithm runs in $O(n^2)$ time.
  - Is there a worst case example where it runs in $\Omega(n^2)$ time?
- What is the expected running time?
- We need to know how often the second case happens.
- We define the indicator random variable X_i:
  - $X_i = 0$ in first case ($v_i = v_{i-1}$).
  - $X_i = 1$ in second case ($v_i \neq v_{i-1}$).
- Then $E[X_i] = P(X_i = 1)$.

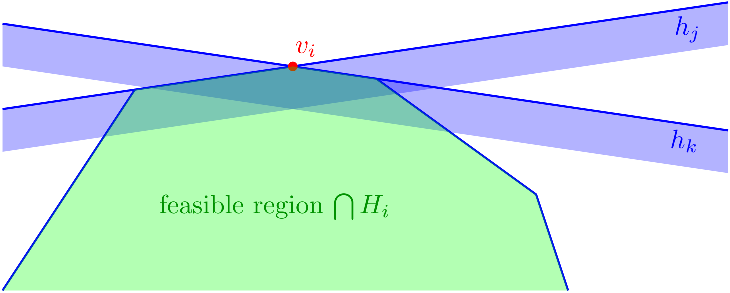
Analysis

- When $X_i = 0$ we spend $O(1)$ time at i-th step.
- When $X_i = 1$ we spend $O(i)$ time.
- So the expected running time $E[T(n)]$ is

$$
\begin{aligned}
E[T(n)] &= O\left(\sum_{i=1}^{n} 1 + i.E[X_i]\right) \\
&= O\left(\sum_{i=1}^{n} 1 + i.P[X_i = 1]\right)
\end{aligned}
$$

Analysis

- The feasible region at step i is $\bigcap H_i$.
- v_i is adjacent to two edges of $\bigcap H_i$, and these edges correspond to two constraints h_j and h_k.



- If $v_i \neq v_{i-1}$, then $i = j$ or $i = k$.

Analysis

- What is the probability that $i = j$ or $i = k$?
- We use backwards analysis.
- We assume that H_i is fixed.
- So h_i is chosen uniformly at random in $\{h_1, h_2, \dots, h_i\}$.
- So the probability that $i = j$ or $i = k$ is at most $2/i$.
  - It could be $1/i$ or 0 if v_i is defined by m_1 and/or m_2.
- So $P[X_i = 1] \leqslant 2/i$, and thus

$$E[T(n)] = O\left(\sum_{i=1}^{n} 1 + i.\frac{2}{i}\right) = O(n).$$

Generalization to higher dimension

First attempt:

- Each constraint is a half-space.
- Can we compute their intersection and get the feasible region?
- In $\mathbb{R}^3$ it can be done in $O(n \log n)$ time. (Not covered in CSE520.)
- In higher dimension, the feasible region has $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ vertices in the worst case.
- So computing the feasible region requires $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$ time.
- Instead of it, we will give a $O(n)$ expected time algorithm for finding one optimal point in the feasible region, when $d = O(1)$.

Preliminary

- A hyperplane in $\mathbb{R}^d$ has equation

$$\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_d x_d = \beta_d.$$

  where

$$(\alpha_1, \alpha_2, \ldots, \alpha_d) \in \mathbb{R}^d \setminus \{0\}^d.$$

- In general position, d hyperplanes intersect at one point.
- Each constraint h_i is a half-space, bounded by a hyperplane ∂h_i. We assume general position in the sense that:
  - Any d hyperplanes $\partial h_{i_1}, \ldots, \partial h_{i_d}$ intersect at exactly one point.
  - The intersection of any $d + 1$ such hyperplanes is empty.
  - No such hyperplane is orthogonal to $\overrightarrow{c}$.

Algorithm

- We generalize the 2D algorithm.
- We first find d constraints $m_1, m_2, \ldots, m_d$ that make the linear program bounded:
  - If $c_i \geqslant 0$ then m_i is $x_i \leqslant M$.
  - If $c_i < 0$ then m_i is $x_i \geqslant -M$.
- We pick a random permutation $(h_1, h_2, \ldots h_n)$ of H.
- Then H_i is $\{m_1, m_2, \ldots m_d, h_1, h_2, \ldots h_i\}$.
- We maintain v_i, the solution to the linear program with constraints H_i and objective function f.
- v_0 is the intersection point

$$\bigcap_{i=1}^{d} \partial m_i.$$

Algorithm

- We assume $d = O(1)$.
- Inserting h_i is done in the same way as in $\mathbb{R}^2$:
- If $v_{i-1} \in h_i$ then $v_{i-1} = v_i$.
- Otherwise, $v_i \in \partial h_i$.
  - We find v_i by solving a linear program with $i + d - 1$ constraints in $\mathbb{R}^{d-1}$.
    - ⋆ If this linear program is infeasible, then the original linear program is infeasible too, so we are done.
  - It can be done in expected $O(i)$ time.
    (By induction.)

Analysis

- What is the probability that $v_i \neq v_{i-1}$?
  - By our general position assumption, v_i belongs to exactly d hyperplanes that bound constraints in H_i.
  - The probability that $v_i \neq v_{i-1}$ is the probability that one of these d constraints was inserted last.
  - By backwards analysis, it is $\leqslant d/i$.
- So the expected running time of our algorithm is

$$E[T(n)] = O\left(\sum_{i=1}^{n} 1 + i.\frac{d}{i}\right) = O(dn) = O(n).$$

Conclusion

- This algorithm can be made to handle unbounded linear programs and degenerate cases.
- A careful implementation of this algorithm runs in $O(d!n)$ time.
- So it is only useful in low dimension.
- It can be generalized to other types of problems.
  - See textbook: Smallest enclosing disk.