Chapter 6: Prune and Search

Chin Lung Lu

Department of Computer Science

National Tsing Hua University

General method of prune and search strategy

- The prune and search strategy always consists of several iterations.
- At each iteratin, it prunes away a fraction f of the input data, where $f < 1$, and uses the same algorithm recursively to solve the problem for the remaining data.
- After $p \in Z^+$ iterations, the size of the remaining data will be q, which is so small that the problem can be solved directly in constant time, say c'.

Running time of prune & search strategy

- Assume that the time needed to execute the purne and search in each iteration is $\mathcal{O}(n^k)$ for some positive constant k.
- Let $T(n)$ be the worst-case running time of prune and search algorithm.

Recursive formula of $T(n)$:

$T(n) = T((1-f)n) + \mathcal{O}(n^k)$, where $1 - f < 1$.

- It can be proved that $T(n) = \mathcal{O}(n^k)$.

Running time of prune & search strategy (cont'd)

For sufficiently large n, we have:

$$
\begin{aligned}
T(n) &\leq T((1-f)n) + cn^k \\
&\leq T((1-f)^2 n) + cn^k + c(1-f)^k n^k \\
&\vdots \\
&\leq c' + cn^k + c(1-f)^k n^k + c(1-f)^{2k} n^k + \ldots + c(1-f)^{(p-1)k} n^k \\
&= c' + cn^k \left(1 + (1-f)^k + (1-f)^{2k} + \ldots + (1-f)^{(p-1)k}\right) \\
&\leq c' + cn^k (1 + (1-f)^k + (1-f)^{2k} + \ldots) \\
&= c' + cn^k \tfrac{1}{1-(1-f)^k} \quad \text{(where } c', c, f \text{ are positive constants)} \\
&= \mathcal{O}(n^k)
\end{aligned}
$$

Property of prune and search strategy

Lemma:
The time complexity of the whole prune and search process is of the same order as the time complexity of the prune and search in each iteration, that is, $\mathcal{O}(n^k)$.

- Note that this lemma holds only when k is a positive constant.

Search problem

Definition:
Given a sorted sequence $a_1 \le a_2 \le \ldots \le a_n$ and an element X, the search problem is to determine whether X is present in this list.

Example:

- Let $(a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (1, 2, 3, 4, 5, 6, 7)$ and $X = 1$.
- Clearly X appears in the given list $(a_1, a_2, a_3, a_4, a_5, a_6, a_7)$.

Binary search algorithm

Input: A sorted sequence $a_1 \le a_2 \le \ldots \le a_n$ and X.
Output: j if $a_j = X$; otherwise, 0.

1. $l = 1$; `/* leftmost index */`
2. $r = n$; `/* rightmost index */`
3. **while** $l \le r$ **do**
4. $\quad m = \lfloor \frac{l+r}{2} \rfloor$; `/* middle index */`
5. $\quad$ **if** $X = a_m$, **then** output m and stop; `/* success */`
6. $\quad$ **if** $X < a_m$, **then** $r = m - 1$;
7. $\quad$ **else** $l = m + 1$;
8. **end while**
9. $j = 0$; `/* failure */`
10. Output j;

Binary search algorithm (cont'd)

Example 1: $(a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (1, 2, 3, 4, 5, 6, 7)$ and $X = 1$

i-th step	a_1	a_2	a_3	a_4	a_5	a_6	a_7	
	1	2	3	4	5	6	7	
$i = 1$	$\uparrow l$			$\uparrow m$			$\uparrow r$	$a_4 \ne X$
$i = 2$	$\uparrow l$	$\uparrow m$	$\uparrow r$					$a_2 \ne X$
$i = 3$	$\uparrow l, m, r$							$a_1 = X$ (success)

Binary search algorithm (cont'd)

Example 2: $(a_1, a_2, a_3, a_4, a_5, a_6, a_7) = (1, 2, 3, 4, 5, 6, 7)$ and $X = 8$

i-th step	a_1	a_2	a_3	a_4	a_5	a_6	a_7	
	1	2	3	4	5	6	7	
$i = 1$	$\uparrow l$			$\uparrow m$			$\uparrow r$	$a_4 \neq X$
$i = 2$					$\uparrow l$	$\uparrow m$	$\uparrow r$	$a_6 \neq X$
$i = 3$							$\uparrow l, m, r$	$a_7 \neq X$ (failure)

Time complexity of binary search algorithm

- Let $T(n)$ be the time complexity of the binary search algorithm.
- Clearly, we have $T(n) = T(\frac{n}{2}) + \mathcal{O}(1)$.
- As a result, $T(n) = \mathcal{O}(\log n)$ by the master theorem.

Selection problem

Definition:
Given a set S of n elements and an integer $1 \leq k \leq n$, the selection problem is to determine the kth smallest element.

Example:

- Let $S = \{5, 3, 7, 1, 9\}$ and $k = 3$.
- The third smallest element in S is 5.

Selection problem (cont'd)

Method 1: Using sorting algorithm

- Sort these n elements in ascending order.
- Locate the kth element.
- The time complexity of this approach is $\mathcal{O}(n \log n)$ time.

Method 2: Using prune and search approach

- The time complexity of this approach is $\mathcal{O}(n)$ time.

Selection problem (cont'd)

Prune and search algorithm

- Let S be the input set of n elements.
- Let $p \in S$.
- Partition S into 3 subsets S_1, S_2 and S_3 such that:
  1. S_1 contains all elements $< p$.
  2. S_2 contains all elements $= p$.
  3. S_3 contains all elements $> p$.

Selection problem (cont'd)

Prune and search algorithm

Case 1:
If $|S_1| \geq k$, the kth smallest element of S is in S_1 and hence we can prune away S_2 and S_3 at the next iteration.

Case 2:
Otherwise, if $|S_1| + |S_2| \geq k$, p is the kth smallest element of S.

Case 3:
If none of above two conditions is satisfied, the kth smallest element of S must be in S_3 and therefore we can discard S_1 and S_2 and start next iteration by selecting the $(k - |S_1| - |S_2|)$th smallest element from S_3.

Selection problem (cont'd)

Prune and search algorithm

Question:
How to select p such that we can always discard a fraction of S?

1. Divide S into $\lceil \frac{n}{5} \rceil$ subsets, with each subset having 5 elements, and add some dummy ∞ elements to the last subset if $n \neq 0 \pmod 5$.
2. Sort each of $\lceil \frac{n}{5} \rceil$ 5-element subsets.
3. Select the median of each subset to form $M = \{m_1, \ldots, m_{\lceil \frac{n}{5} \rceil}\}$ and let p be the median of M.

Selection problem (cont'd)

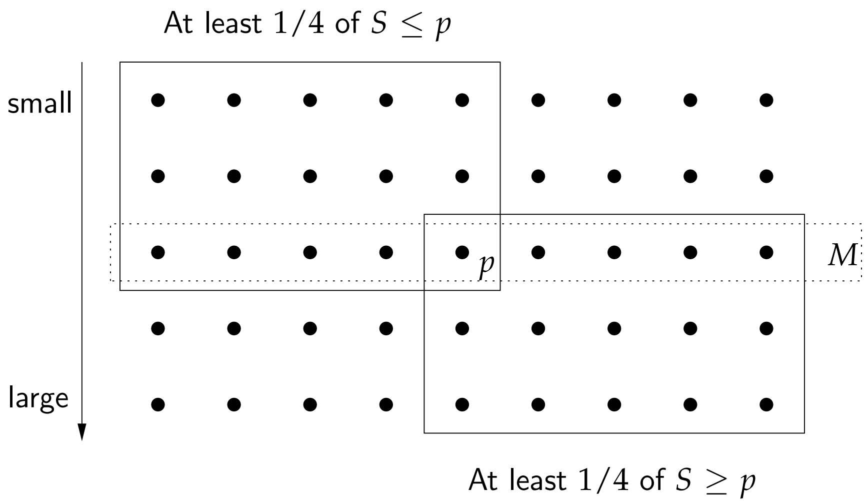
Prune and search algorithm

Lemma (also refer to the figure on next slide):
At least $1/4$ of the elements in S are $\leq p$ and at least $1/4$ of the elements in S are $\geq p$.

- Hence, if we choose p in this way, we can always prune away at least $\frac{1}{4} \times |S|$ elements from S during each iteration.

Selection problem (cont'd)

Prune and search algorithm



Selection problem (cont'd)

Prune and search algorithm

Input: A set S of n elements.
Output: The kth smallest element of S.

1. If $|S| < 5$, solve the problem by any brute force method.
2. Divide S into $\lceil \frac{n}{5} \rceil$ subsets, with each subset containing five elements, and add some dummy ∞ elements to the last subset when $n \neq 0 \pmod 5$.
3. Sort each of $\lceil \frac{n}{5} \rceil$ 5-element subsets.
4. Find p which is the median of the medians of the $\lceil \frac{n}{5} \rceil$ subsets.
5. Partition S into three subsets S_1, S_2 and S_3, which contains the elements less than, equal to, and greater than p, respectively.

Selection problem (cont'd)

Prune and search algorithm

6. **if** $|S_1| \geq k$ **then**
   Discard S_2 and S_3 and solve the problem recursively.
   else if $|S_1| + |S_2| \geq k$ **then**
   p is the kth smallest element of S.
   else
   Discard S_1 and S_2 and solve the problem by selecting $(k - |S_1| - |S_2|)$th element at the next iteration.
   end if
   end if

Selection problem (cont'd)

Time complexity of prune and search algorithm

- Let $T(n)$ be the time of the above prune and search algorithm.
- Steps 1, 2, 3 and 5 cost $\mathcal{O}(n)$ time.
- Step 4 costs $T(\lceil \frac{n}{5} \rceil)$ if we use the same algorithm recursively to find the median of the $\lceil \frac{n}{5} \rceil$ elements.
- Because we always prune away at least $\frac{1}{4}$ elements during each iteration, the problem remaining in step 6 contains at most $\frac{3}{4}$ elements and hence can be accomplished in $T(\frac{3n}{4})$ time.
- As a result, we have:

$$T(n) = T\left(\frac{3n}{4}\right) + T\left(\frac{n}{5}\right) + \mathcal{O}(n).$$

Selection problem (cont'd)

Time complexity of prune and search algorithm

- Let $T(n) = a_0 + a_1 n + a_2 n^2 + \cdots$.
- Then we have:

$$T\left(\frac{3n}{4}\right) = a_0 + \frac{3}{4}a_1 n + \frac{9}{16}a_2 n^2 + \cdots$$

$$T\left(\frac{n}{5}\right) = a_0 + \frac{1}{5}a_1 n + \frac{1}{25}a_2 n^2 + \cdots$$

$$T\left(\frac{3n}{4} + \frac{n}{5}\right) = T\left(\frac{19n}{20}\right) = a_0 + \frac{19}{20}a_1 n + \frac{361}{400}a_2 n^2 + \cdots$$

- Therefore, we have $T\left(\frac{3n}{4}\right) + T\left(\frac{n}{5}\right) \leq a_0 + T\left(\frac{19n}{20}\right)$.

Selection problem (cont'd)

Time complexity of prune and search algorithm

- Hence, the time complexity of the prune and search algorithm for the selection problem is:

$$\begin{aligned} T(n) &= T\left(\frac{3n}{4}\right) + T\left(\frac{n}{5}\right) + \mathcal{O}(n) \\ &\leq T\left(\frac{19n}{20}\right) + cn \\ &= \mathcal{O}(n) \end{aligned}$$

- As a result, the selection problem can be solved by the prune and search algorithm in linear time.

Linear programming problem

Definition:

$$\begin{array}{ll} \text{Minimize} & c_1x_1 + c_2x_2 + \ldots + c_nx_n \\ \text{Subject to} & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \geq b_2 \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \geq b_m \end{array}$$

- The problem is to optimize a linear function of several variables satisfying some constraints in the form of linear equalities and inequalities.

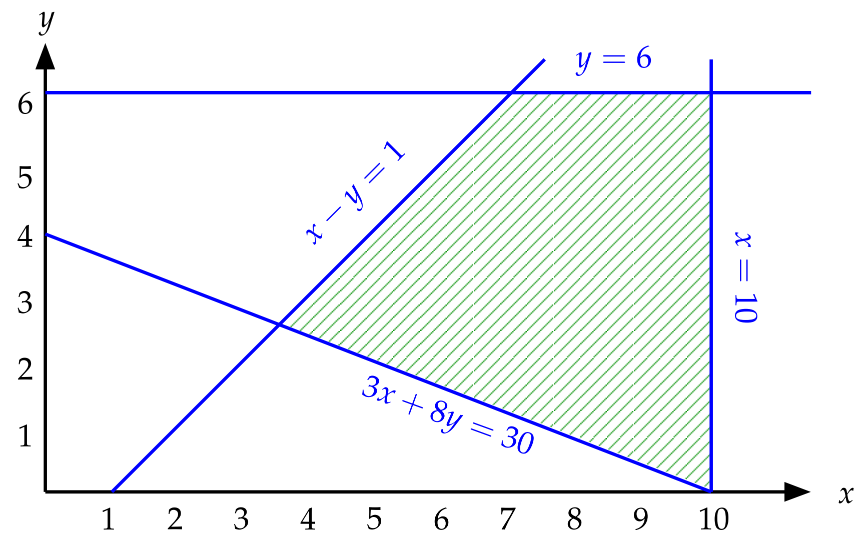
Linear programming problem (cont'd)

Example:

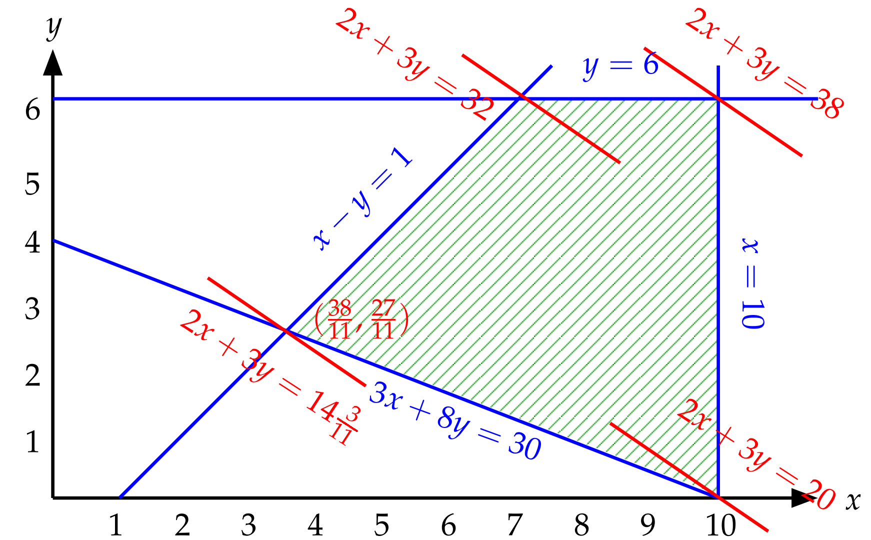
$$\begin{array}{ll} \text{Minimize} & 2x + 3y \\ \text{Subject to} & x \leq 10 \\ & y \leq 6 \\ & x - y \geq 1 \\ & 3x + 8y \geq 30 \end{array}$$

- The optimum solution is located at $(x, y) = (\frac{38}{11}, \frac{27}{11})$.

Linear programming problem (cont'd)



Linear programming problem (cont'd)



Linear programming problem (cont'd)

- Megiddo (1984) and Dyer (1984) independently designed a prune and search strategy to solve the linear programming problem with a fixed number of variables in $\mathcal{O}(n)$ time, where n is the number of constraints.
- Below, we will introduce the prune and search technique to solve the linear programming problem with two variables.

Two-variable linear programming problem

Definition:
Minimize $ax + by$
Subject to $a_i x + b_i y \geq c_i$ for $i = 1, 2, \ldots, n$

Two-variable linear programming problem

Basic idea of prune and search method

- There are always some constraints that have nothing to do with the solution and hence can be pruned away.
- In the prune and search method, a fraction of constraints are pruned away after every iteration.
- After several iterations, the number of the constraints will be so small that the linear programming problem can be solved in some constant time.

Simplified two-variable LP problem

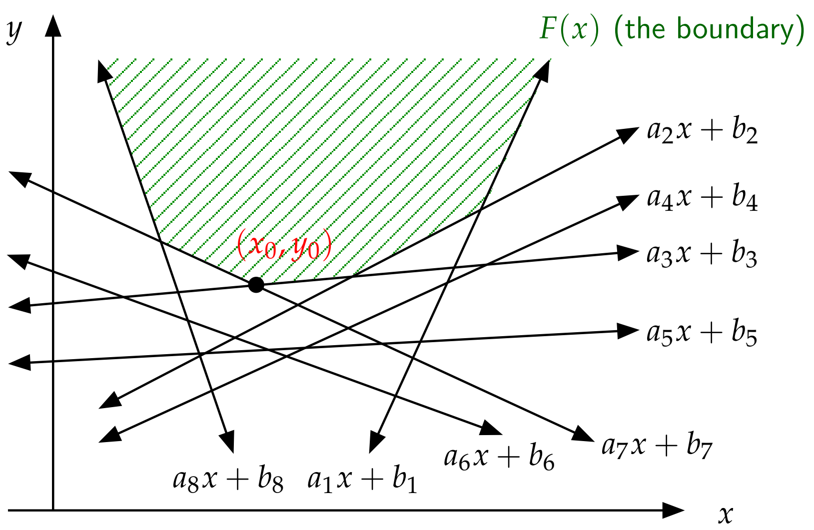
- To simplify the discussion, we consider the simplified version of the two-variable linear programming problem.

Simplified two-variable linear programming problem:

Minimize y

Subject to $y \geq a_i x + b_i$ for $i = 1, 2, \ldots, n$

Simplified two-variable LP problem (cont'd)



(x_0, y_0) is the optimum solution.

Simplified two-variable LP problem (cont'd)

- Because $y \geq a_i x + b_i$ for all i, the optimal solution must be on the boundary surrounding the feasible region.
- Let $F(x) = \max_{1 \leq i \leq n} \{a_i x + b_i\}$ (the boundary of feasible region).
- Note that for each x, $F(x)$ must have the highest value among all n constraints.

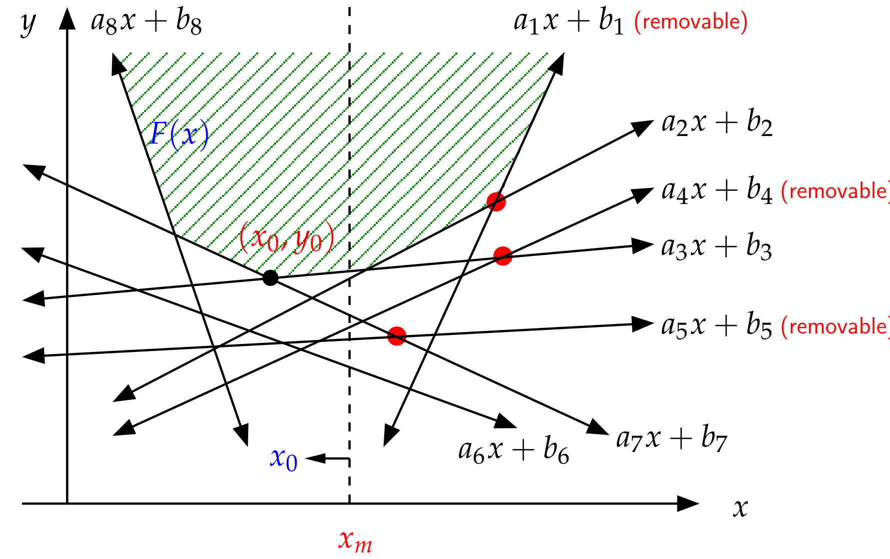
Simplified two-variable LP problem (cont'd)

Observation:
The optimum solution x_0 satisfies $F(x_0) = \min_{-\infty \leq x \leq \infty} F(x)$.

Assumptions:

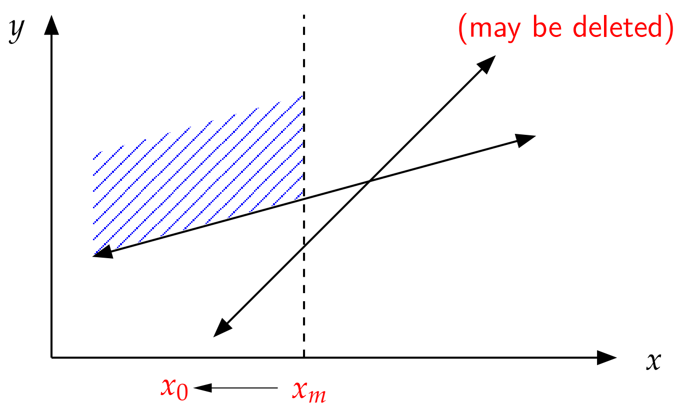
1. We have picked a point x_m on the x-axis as shown in the figure on the next slide.
2. By some reasoning, we know that $x_0 \leq x_m$ (or $x_0 \geq x_m$).

Simplified two-variable LP problem (cont'd)



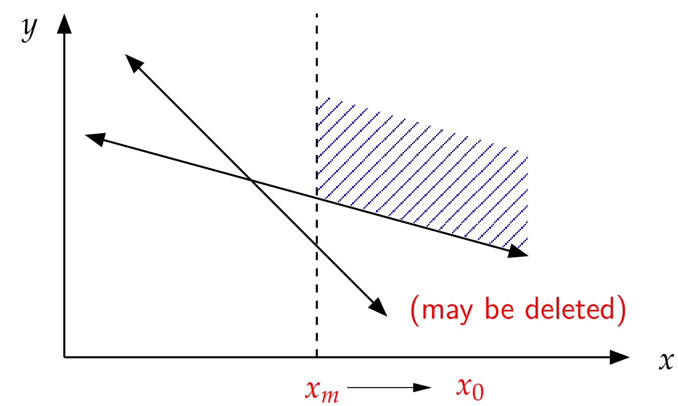
Simplified two-variable LP problem (cont'd)

Case 1:
If $x_0 < x_m$ and the intersection of two constraints is to the right of x_m, one of these two constraints is always smaller than the other for $x < x_m$ and hence this constraint may be deleted.



Simplified two-variable LP problem (cont'd)

Case 2:
If $x_0 > x_m$ and the intersection of two constraints is to the left of x_m, one of these two constraints is always smaller than the other for $x > x_m$ and hence this constraint may be deleted.



Simplified two-variable LP problem (cont'd)

Question:

Suppose that x_m is known. How do we know whether $x_0 < x_m$ or $x_0 > x_m$?

- Let $y_m = F(x_m) = \max_{1 \le i \le n} \{a_i x_m + b_i\}$.
- Obviously, (x_m, y_m) is a point on the boundary of the feasible region.

Simplified two-variable LP problem (cont'd)

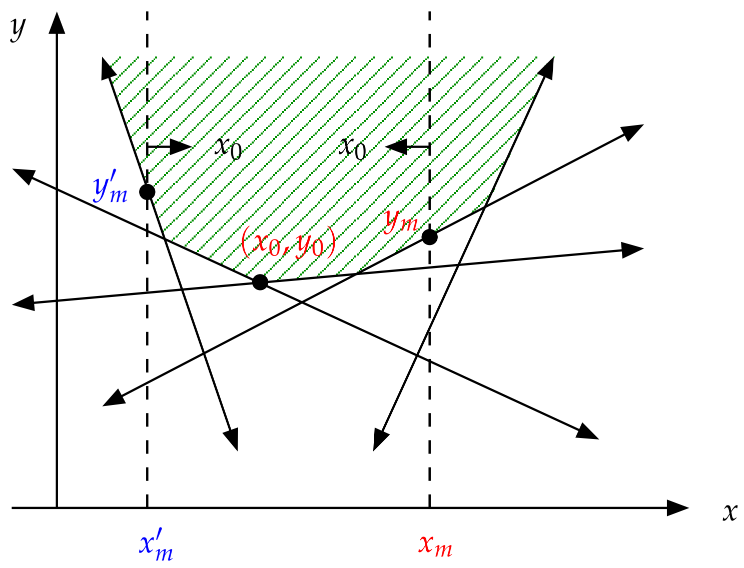
How do we know whether $x_0 < x_m$ or $x_0 > x_m$?

Case 1: y_m is on only one constraint

- Let g be the slope of this constraint.
- If $g > 0$ then $x_0 < x_m$.
- If $g < 0$, then $x_0 > x_m$.

Simplified two-variable LP problem (cont'd)

How do we know whether $x_0 < x_m$ or $x_0 > x_m$?



Simplified two-variable LP problem (cont'd)

How do we know whether $x_0 < x_m$ or $x_0 > x_m$?

Case 2: y_m is the intersection of several constraints

- Let $g_{\max}$ and $g_{\min}$ denote the maximum and minimum slopes of these constraints respectively.

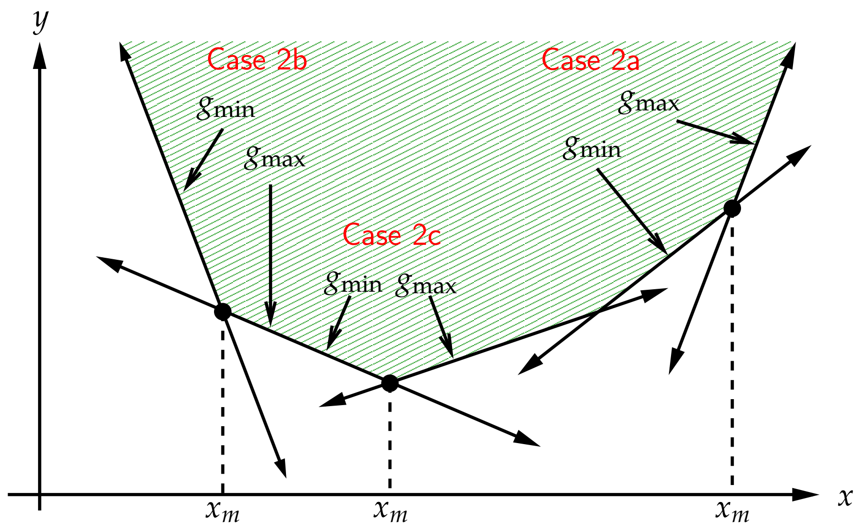
$$g_{\max} = \max_{1 \le i \le n} \{a_i | a_i x_m + b_i = F(x_m)\}$$

$$g_{\min} = \min_{1 \le i \le n} \{a_i | a_i x_m + b_i = F(x_m)\}$$

Case 2a: If $g_{\max} > 0$ and $g_{\min} > 0$, then $x_0 < x_m$.
Case 2b: If $g_{\max} < 0$ and $g_{\min} < 0$, then $x_0 > x_m$.
Case 2c: If $g_{\max} > 0$ and $g_{\min} < 0$, then (x_m, y_m) is the optimum solution.

Simplified two-variable LP problem (cont'd)

How do we know whether $x_0 < x_m$ or $x_0 > x_m$?



Simplified two-variable LP problem (cont'd)

Question:

How are we going to choose x_m?

- For n constraints, group $\frac{n}{2}$ pairs of constraints and find their intersections.
- The x_m should be chosen so that half of the intersections lie to the right of x_m and half of the intersections lie to the left of x_m.
- In other words, x_m is the median of the x-coordinates of $\frac{n}{2}$ intersections.

Simplified two-variable LP problem (cont'd)

Prune and search algorithm

Input: A set S of n constraints $a_i x + b_i$, where $i = 1, 2, \ldots, n$.
Output: The value x_0 such that y is minimized at x_0 subject to $y \geq a_i x + b_i$, where $i = 1, 2, \ldots, n$.

1. If S contains no more than 2 constraints, then solve this problem by a brute force method.
2. Divide S into $\frac{n}{2}$ pairs of constraints.
   For each pair of $a_i x + b_i$ and $a_j x + b_j$, find the intersection p_{ij} of them and denote its x-value as x_{ij}.
3. Find the median x_m among the x_{ij}'s (at most $\frac{n}{2}$ of them).
4. Determine $y_m = F(x_m) = \max_{1 \leq i \leq n}\{a_i x_m + b_i\}$.
   Let $g_{\max} = \max_{1 \leq i \leq n}\{a_i | a_i x_m + b_i = F(x_m)\}$.
   Let $g_{\min} = \min_{1 \leq i \leq n}\{a_i | a_i x_m + b_i = F(x_m)\}$.

Simplified two-variable LP problem (cont'd)

Prune and search algorithm

5. **if** $g_{\max}$ and $g_{\min}$ are not of the same sign **then**
   x_m is the solution and exit.
   else $x_0 < x_m$ if $g_{\min} > 0$, and $x_0 > x_m$ if $g_{\min} < 0$.
6. **if** $x_0 < x_m$ **then**
   for each pair of $a_i x + b_i$ and $a_j x + b_j$ whose $x_{ij} > x_m$ **do**
   Prune away the constraint which is smaller than the other for $x \leq x_m$.
   if $x_0 > x_m$ **then**
   for each pair of $a_i x + b_i$ and $a_j x + b_j$ whose $x_{ij} < x_m$ **do**
   Prune away the constraint which is smaller than the other for $x \geq x_m$.
   Let S denote the set of remaining constraints and go to step 1.

Simplified two-variable LP problem (cont'd)

Time complexity of prune and search algorithm

- Step 2 costs $\mathcal{O}(n)$ time since the intersection of two lines can be found in constant time.
- Step 3 takes $\mathcal{O}(n)$ time because the median can be found in $\mathcal{O}(n)$ time.
- Step 4 costs $\mathcal{O}(n)$ time by scanning all of the constraints.
- Step 5 takes constant time.
- Steps 6 costs $\mathcal{O}(n)$ time by scanning all intersecting pairs.

Simplified two-variable LP problem (cont'd)

Time complexity of prune and search algorithm

- There are $\lfloor \frac{n}{4} \rfloor$ constraints that can be pruned away, since there are $\lfloor \frac{n}{2} \rfloor$ intersections and $\frac{1}{2} \times \lfloor \frac{n}{2} \rfloor$ constraints are pruned away for each iteration.
- The time complexity $T(n)$ of the prune and search algorithm is:

$$T(n) = T\left(\frac{3n}{4}\right) + \mathcal{O}(n) = \mathcal{O}(n)$$

Two-variable linear programming problem

- We return to original 2-variable linear programming problem.

Definition:

Minimize $ax + by$

Subject to $a_i x + b_i y \geq c_i \quad (i = 1, 2, \ldots, n)$

Example:

Minimize $2x + 3y$

Subject to $x \leq 10$

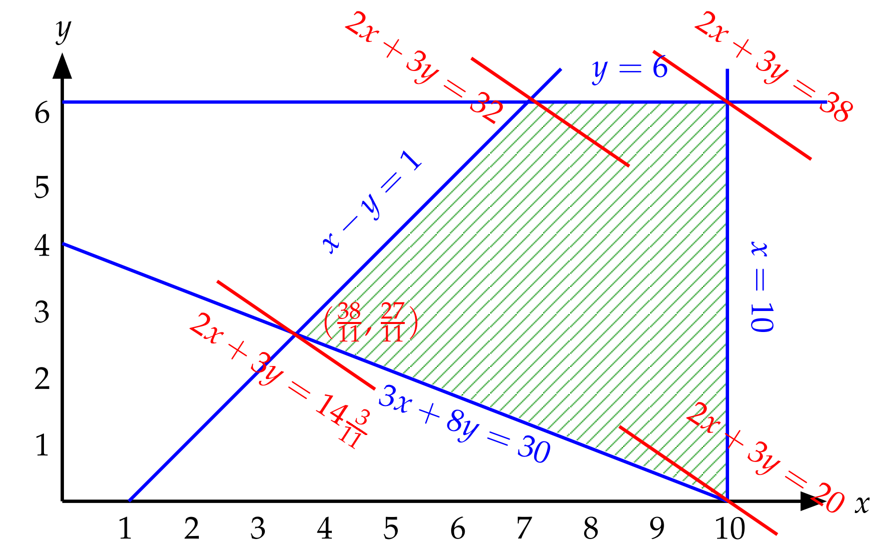
$y \leq 6$

$x - y \geq 1$

$3x + 8y \geq 30$

- The optimum solution is located at $(x, y) = (\frac{38}{11}, \frac{27}{11})$.

Two-variable linear programming (cont'd)



Two-variable linear programming (cont'd)

Original two-variable linear programming problem:

Minimize $ax + by$

Subject to $a_i x + b_i y \geq c_i \quad (i = 1, 2, \ldots, n)$

- We can transform the two-variable linear programming problem into a problem of minimizing the y-value only.

Transformed two-variable linear programming problem:

Minimize y'

Subject to $a'_i x' + b'_i y' \geq c'_i \quad (i = 1, 2, \ldots, n)$

where $a'_i = a_i - \frac{b_i a}{b}, b'_i = \frac{b_i}{b}$ and $c'_i = c_i$

Two-variable linear programming (cont'd)

- Let $x' = x$ and $y' = ax + by$.
- We have $x = x'$ and $y = \frac{y' - ax'}{b}$.
- Furthermore, the constraint $a_i x + b_i y \geq c_i$ can be rewritten as $a'_i x' + b'_i y' \geq c'_i$, where $a'_i = a_i - \frac{b_i a}{b}, b'_i = \frac{b_i}{b}$ and $c'_i = c_i$.

$$
\begin{aligned}
& a_i x + b_i y \geq c_i \\
\Rightarrow\ & a_i x' + b_i \frac{y' - ax'}{b} \geq c_i \\
\Rightarrow\ & (a_i - \frac{b_i a}{b}) x' + \frac{b_i}{b} y' \geq c_i
\end{aligned}
$$

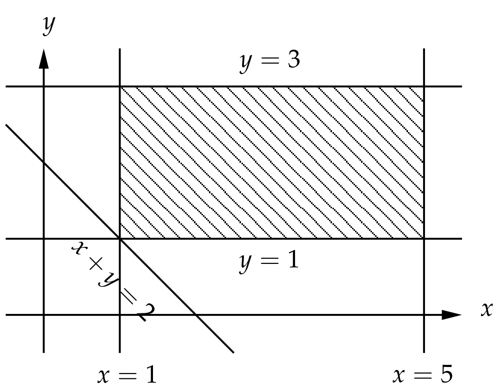
Two-variable linear programming (cont'd)

Minimize $x + y$

Subject to $y \geq 1$

$y \leq 3$

$1 \leq x \leq 5$



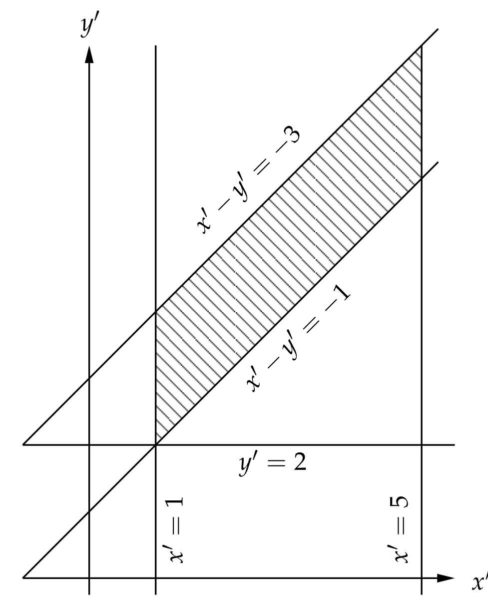
Two-variable linear programming (cont'd)

Minimize y'

Subject to $x' - y' \leq -1$

$x' - y' \geq -3$

$1 \leq x' \leq 5$



Two-variable linear programming (cont'd)

- Therefore, a general two-variable linear programming problem

$$\begin{array}{ll} \text{Minimize} & ax+by \\ \text{Subject to} & a_i x + b_i y \geq c_i \quad (i=1,2,\ldots,n) \end{array}$$

can be transformed to the following problem:

$$\begin{array}{ll} \text{Minimize} & y \\ \text{Subject to} & a_i x + b_i y \geq c_i \quad (i=1,2,\ldots,n) \end{array}$$

- Recall that simplified 2-variable linear programming problem is defined as:

$$\begin{array}{ll} \text{Minimize} & y \\ \text{Subject to} & y \geq a_i x + b_i \quad (i=1,2,\ldots,n) \end{array}$$

Two-variable linear programming (cont'd)

- In general, there are three kinds of constraints when we rewrite $a_i x + b_i y \geq c_i$:

  Case 1: $y \geq a_i x + b_i$

  Case 2: $y \leq a_i x + b_i$

  Case 3: $a \leq x \leq b$
- Thus, we can group the constraints with positive (respectively, negative) coefficients for y in set I_1 (respectively, I_2).

The two-variable linear programming problem becomes:

$$\begin{array}{ll} \text{Minimize} & y \\ \text{Subject to} & y \geq a_i x + b_i \quad (i \in I_1) \\ & y \leq a_i x + b_i \quad (i \in I_2) \\ & a \leq x \leq b \end{array}$$

Two-variable linear programming (cont'd)

Example:

Consider the following four constraints:

$x \leq 10$
$y \leq 6$
$x - y \geq 1$
$3x + 8y \geq 30$

- We can rewrite the above four constraints as follows:

$x \leq 10$
$y \leq 6$ (this constraint belongs to I_2)
$y \leq x - 1$ (this constraint belongs to I_2)
$y \geq \frac{-3x}{8} + \frac{30}{8}$ (this constraint belongs to I_1)

Two-variable linear programming (cont'd)

- Let us define two functions:

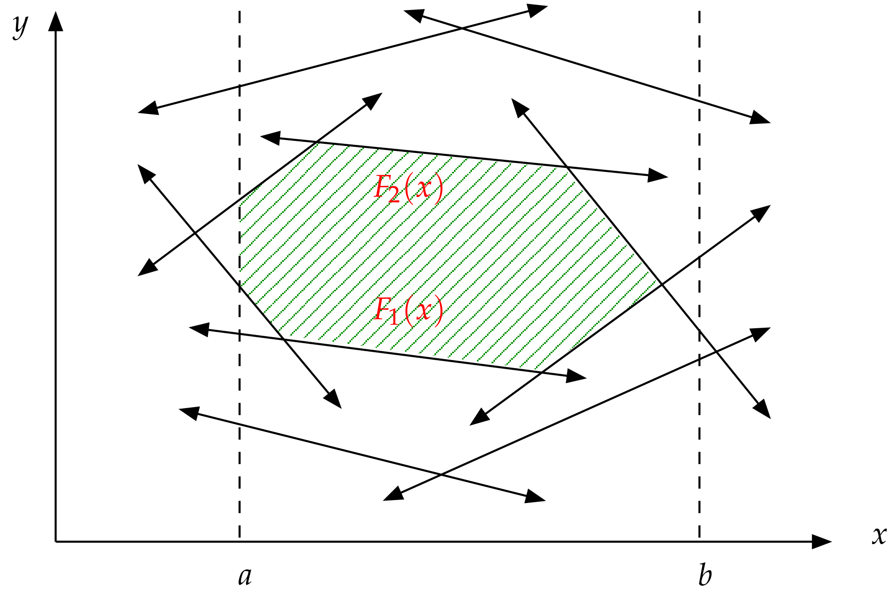
$$\begin{aligned} F_1(x) &= \max\{a_i x + b_i : i \in I_1\} \\ F_2(x) &= \min\{a_i x + b_i : i \in I_2\} \end{aligned}$$

- Then $F_1(x)$, $F_2(x)$ and $a \leq x \leq b$ define the feasible region of the linear programming problem.

The two-variable LP problem can be further transformed to:

$$\begin{array}{ll} \text{Minimize} & F_1(x) \\ \text{Subject to} & F_1(x) \leq F_2(x) \\ & a \leq x \leq b \end{array}$$

Two-variable linear programming (cont'd)



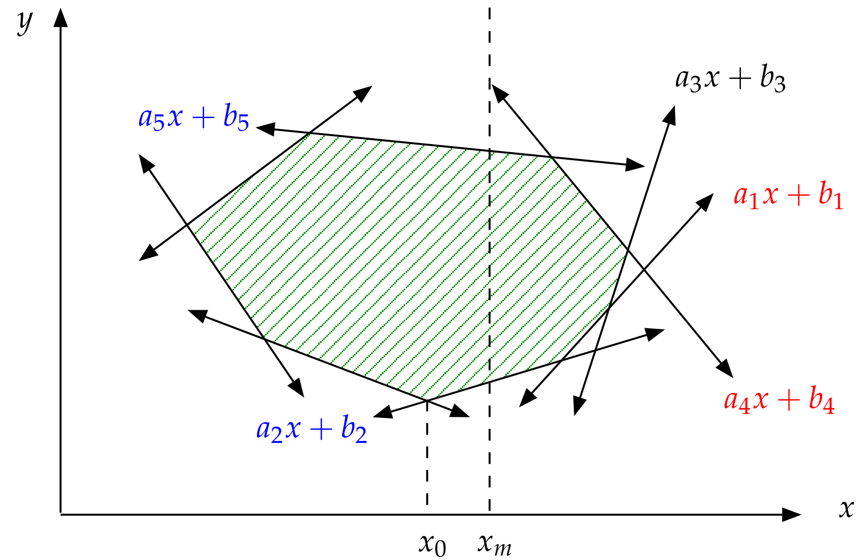
Prune and search algorithm

Two-variable linear programming problem

- Actually, some constraints may be pruned away if we know the searching direction.
- The reason is the same as that used in the special two-variable linear programming problem.
- Example Consider the figure on the next slide.
- If the optimal solution x_0 lies to the left of x_m, $a_1x + b_1$ can be eliminated without affecting the solution.
- The reason is that $a_1x + b_1 < a_2x + b_2$ for $x < x_m$.
- Similarly, $a_4x + b_4$ can be eliminated.
- The reason is that $a_4x + b_4 > a_5x + b_5$ for $x < x_m$.

Prune and search algorithm (cont'd)

Two-variable linear programming problem



Prune and search algorithm (cont'd)

Some questions to be addressed

Given a point x_m, where $a \leq x_m \leq b$, we must decide the following problems:

1. Is x_m feasible?
2. If x_m is feasible, we must decide if the optimum solution x_0 lies to the left or right of x_m.

   It may also happen that that x_m itself may be the optimum solution.
3. If x_m is not feasible, we must decide if there exists any feasible solution or not.

   If a feasible solution exists, we have to decide which side of x_m this optimum solution exists.

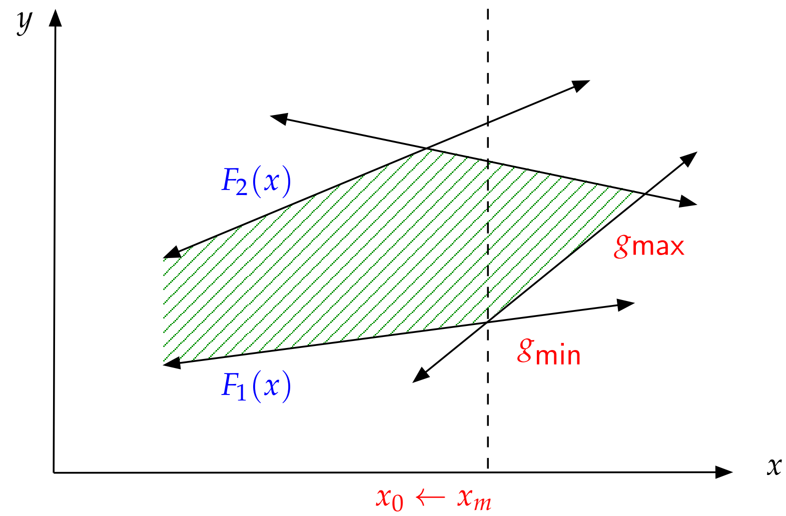
Is x_m feasible?

- Let $F(x) = F_1(x) - F_2(x)$.
- Obviously, x_m is feasible if and only if $F(x_m) \leq 0$.
- Let $g_{\text{min}} = \min\{a_i : i \in I_1, a_i x_m + b_i = F_1(x_m)\}$.
- Let $g_{\text{max}} = \max\{a_i : i \in I_1, a_i x_m + b_i = F_1(x_m)\}$.
- Let $h_{\text{min}} = \min\{a_i : i \in I_2, a_i x_m + b_i = F_2(x_m)\}$.
- Let $h_{\text{max}} = \max\{a_i : i \in I_2, a_i x_m + b_i = F_2(x_m)\}$.

Case 1: $F(x_m) \leq 0$

x_m is feasible

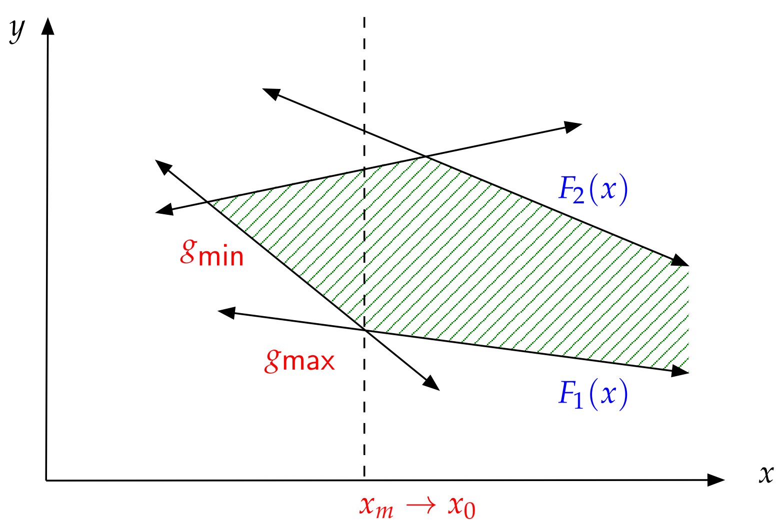
Case 1.1: If $g_{\text{min}} > 0$ and $g_{\text{max}} > 0$, then $x_0 < x_m$.



Case 1: $F(x_m) \leq 0$ (cont'd)

x_m is feasible

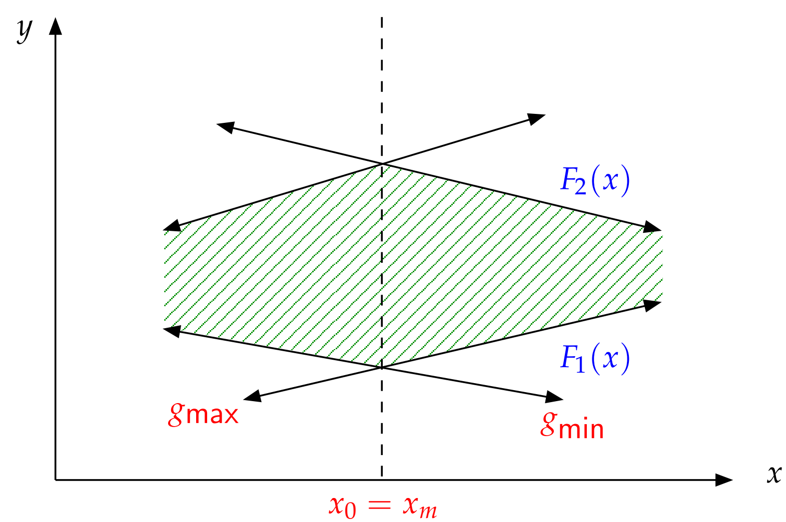
Case 1.2: If $g_{\text{min}} < 0$ and $g_{\text{max}} < 0$, then $x_0 > x_m$.



Case 1: $F(x_m) \leq 0$ (cont'd)

x_m is feasible

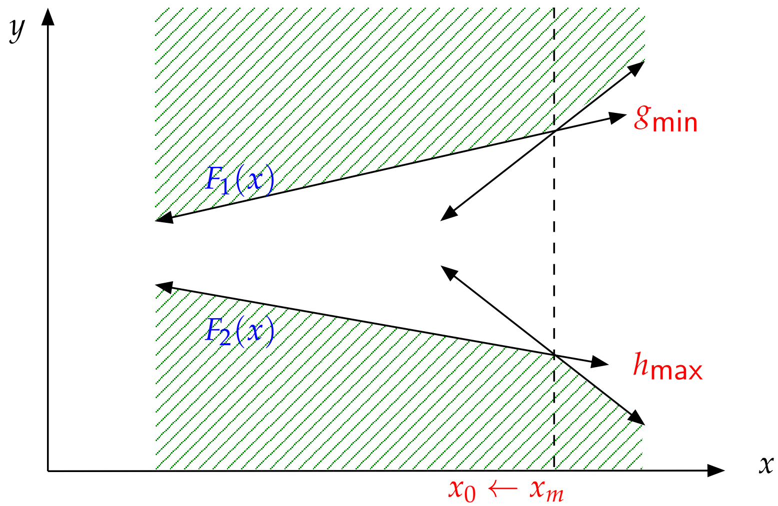
Case 1.3: If $g_{\text{min}} < 0$ and $g_{\text{max}} > 0$, then $x_m = x_0$.



Case 2: $F(x_m) > 0$

x_m is infeasible

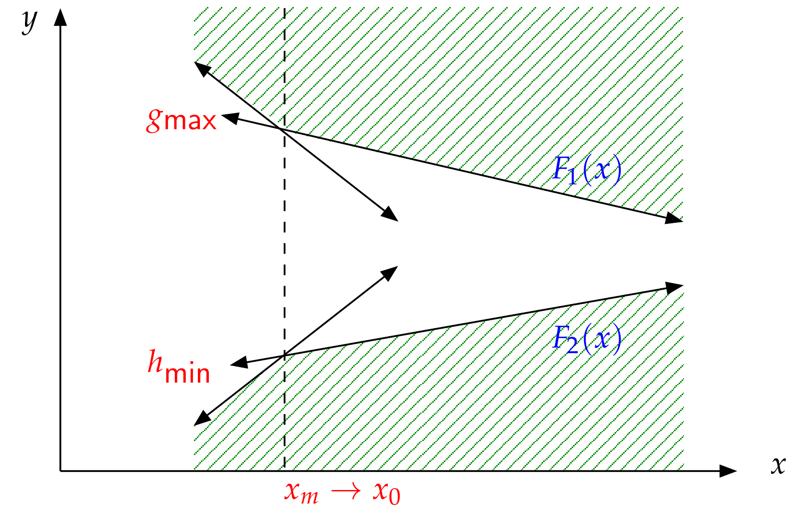
Case 2.1: If $g_{\min} > h_{\max}$, then $x_0 < x_m$.



Case 2: $F(x_m) > 0$ (cont'd)

x_m is infeasible

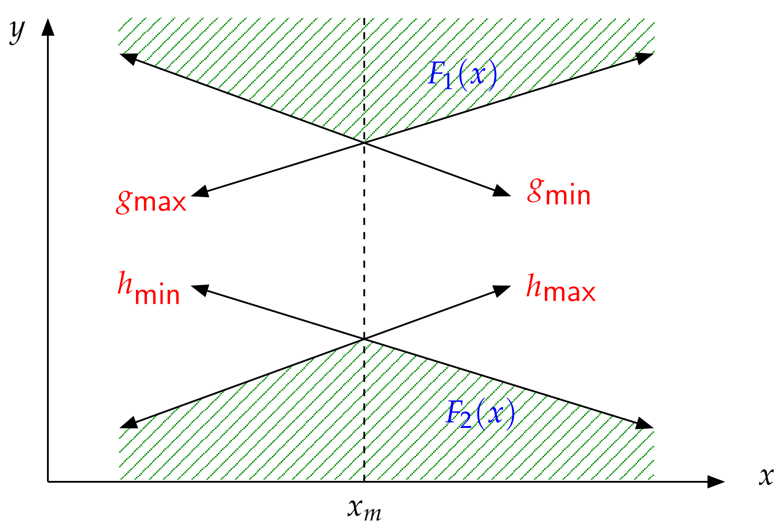
Case 2.2: If $g_{\max} < h_{\min}$, then $x_0 > x_m$.



Case 2: $F(x_m) > 0$ (cont'd)

x_m is infeasible

Case 2.3: If $g_{\min} \leq h_{\max}$ and $g_{\max} \geq h_{\min}$, no feasible solution exists.



A procedure to address some questions

Procedure 6-1

Input: The value x_m of x-coordinate of a point.

Output: Whether it is meaningful to continue searching from x_m and if yes, output the direction of the searching.

1. $F_1(x) = \max\{a_i x + b_i : i \in I_1\}$.
   $F_2(x) = \min\{a_i x + b_i : i \in I_2\}$.
   $F(x) = F_1(x) - F_2(x)$.
2. $g_{\min} = \min\{a_i : i \in I_1, a_i x_m + b_i = F_1(x_m)\}$.
   $g_{\max} = \max\{a_i : i \in I_1, a_i x_m + b_i = F_1(x_m)\}$.
   $h_{\min} = \min\{a_i : i \in I_2, a_i x_m + b_i = F_2(x_m)\}$.
   $h_{\max} = \max\{a_i : i \in I_2, a_i x_m + b_i = F_2(x_m)\}$.

A procedure to address some questions (cont'd)

Procedure 6-1 (cont'd)

3. **Case 1:** $F(x_m) \leq 0$ `/* `x_m` is feasible */`
   (a) If $g_{\text{min}} > 0$ and $g_{\text{max}} > 0$, report $x_0 < x_m$ and exit.
   (b) If $g_{\text{min}} < 0$ and $g_{\text{max}} < 0$, report $x_0 > x_m$ and exit.
   (c) If $g_{\text{min}} < 0$ and $g_{\text{max}} > 0$, report "x_m is the optimal solution" and exit.

   Case 2: $F(x_m) > 0$ `/* `x_m` is infeasible */`
   (a) If $g_{\text{min}} > h_{\text{max}}$, report $x_0 < x_m$ and exit.
   (b) If $g_{\text{max}} < h_{\text{min}}$, report $x_0 > x_m$ and exit.
   (c) If $g_{\text{min}} \leq h_{\text{max}}$ and $g_{\text{max}} \geq h_{\text{min}}$, report "no feasible solution exists" and exit.

Prune and search algorithm

A algorithm for solving two-variable linear programming problem

1. If there are no more than two constraints in I_1 and I_2, solve this problem by a brute force method.
2. Arrange the constraints in I_1 into disjoint pairs, as well as the constraints in I_2 into disjoint pairs.
   For each pair, if $a_ix + b_i$ is parallel to $a_jx + b_j$, delete $a_ix + b_i$ if $b_i < b_j$ for $i, j \in I_1$ or $b_i > b_j$ for $i, j \in I_2$.
   Otherwise, find the intersection p_{ij} of $y = a_ix + b_i$ and $y = a_jx + b_j$ and let the x-coordinate of p_{ij} be x_{ij}.
3. Find the median x_m of x_{ij}'s.
4. Apply Procedure 6-1 to x_m.
   If x_m is the optimal, report x_m and exit.
   If no feasible solution exits, report this and exit.

Prune and search algorithm (cont'd)

5. **if** $x_0 > x_m$ **then**
   For each $x_{ij} < x_m$ and $i, j \in I_1$, prune constraint $y \geq a_ix + b_i$ if $a_i < a_j$; otherwise, prune $y \geq a_jx + b_j$.
   For each $x_{ij} < x_m$ and $i, j \in I_2$, prune constraint $y \leq a_ix + b_i$ if $a_i > a_j$; otherwise, prune $y \leq a_jx + b_j$.
   end if
   if $x_0 < x_m$ **then**
   For each $x_{ij} > x_m$ and $i, j \in I_1$, prune constraint $y \geq a_ix + b_i$ if $a_i > a_j$; otherwise, prune $y \geq a_jx + b_j$.
   For each $x_{ij} > x_m$ and $i, j \in I_2$, prune constraint $y \leq a_ix + b_i$ if $a_i < a_j$; otherwise, prune $y \leq a_jx + b_j$.
   end if
6. Go to step 1.

Time complexity of prune and search algorithm

- Because we can prune $\lfloor \frac{n}{4} \rfloor$ constraints for each iteration, and each step in the prune and search algorithm takes $\mathcal{O}(n)$ time, the algorithm is of the order $\mathcal{O}(n)$.

Constrained 1-center problem

- Below, we introduce an algorithm for the so-called constrained 1-center problem where the center is restricted to lying on a straight line.
- Without losing generality, we assume that the straight line is $y = y'$.

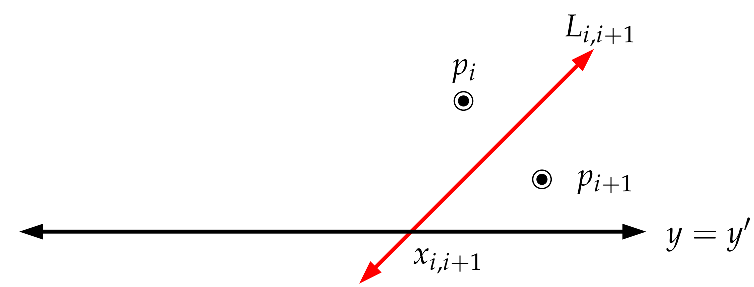
Definition:

- Input: n planar points and a straight line $y = y'$
- Output: a smallest circle to cover the n planar points such that its center (x^*, y') lies on $y = y'$

Prune and search algorithm

Constrained 1-center problem

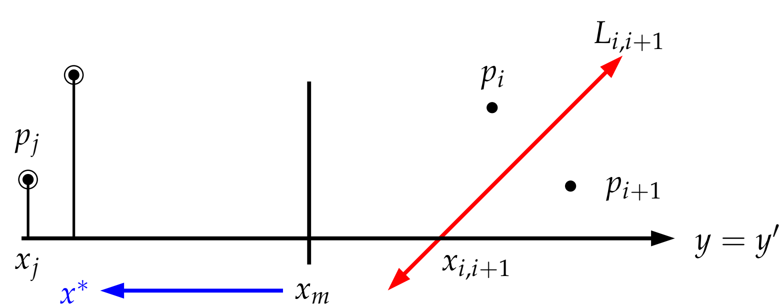
1. If $n \leq 2$, solve this problem by a brute force method.
2. Form disjoint pairs of points $(p_1, p_2), \ldots, (p_{n-1}, p_n)$.
   If there are odd number points, let the final pair be (p_n, p_1).
3. For each pair (p_i, p_{i+1}), find the point $x_{i,i+1}$ on $y = y'$ such that $d(p_i, x_{i,i+1}) = d(p_{i+1}, x_{i,i+1})$.



Prune and search algorithm (cont'd)

Constrained 1-center problem

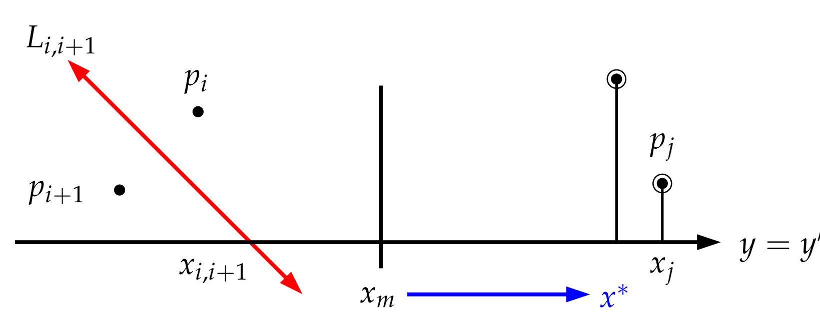
4. Find the median of the $\lceil \frac{n}{2} \rceil$ $x_{i,i+1}$'s and denote it by x_m.
5. Calculate the distance between p_i and x_m for all i.
   Let $I = \{p_j : p_j$ is a farthest point from $x_m\}$.
   Let x_j be the projection of p_j on $y = y'$ for each $p_j \in I$.
   Case 1: If $x_j < x_m$ for every $p_j \in I$, then $x^* < x_m$.



Prune and search algorithm (cont'd)

Constrained 1-center problem

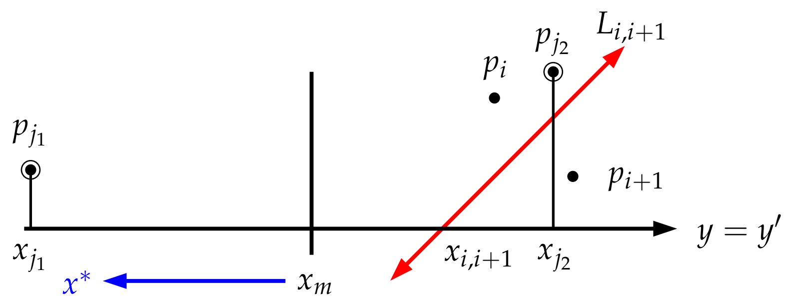
5. (cont'd)
   Case 2: If $x_j > x_m$ for every $p_j \in I$, then $x^* > x_m$.



Prune and search algorithm (cont'd)

Constrained 1-center problem

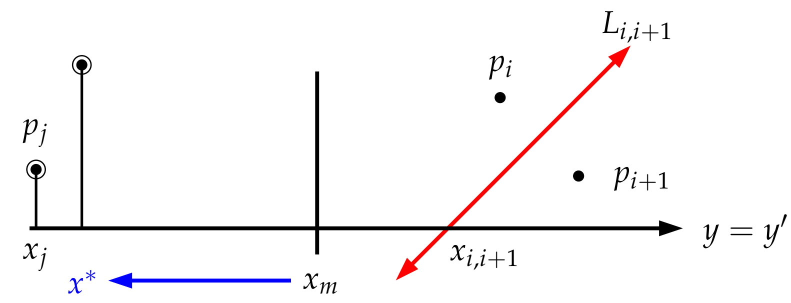
5. (cont'd)
   Case 3: For other cases, we have $x^* = x_m$.
   /* Note that $p_{j_1}, p_{j_2} \in I$ */



Prune and search algorithm (cont'd)

Constrained 1-center problem

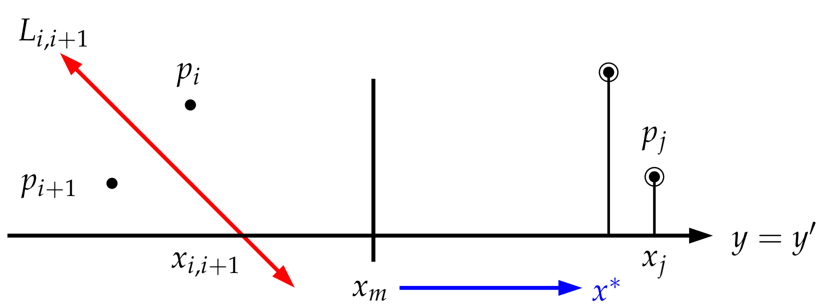
6. **if** $x^* < x_m$ **then**
   For each $x_{i,i+1} > x_m$, prune away p_i if p_i is closer to x_m than p_{i+1}; otherwise, prune away p_{i+1}.
   end if



Prune and search algorithm (cont'd)

Constrained 1-center problem

6. (cont'd)
   if $x^* > x_m$ **then**
   For each $x_{i,i+1} < x_m$, prune away p_i if p_i is closer to x_m than p_{i+1}; otherwise, prune away p_{i+1}.
   end if
7. Go to step 1.



Time complexity of prune and search algorithm

Constrained 1-center problem

- There are $\lfloor \frac{n}{4} \rfloor$ $x_{i,i+1}$'s lying in the left (right) side of x_m.
- Hence, we can prune away $\lfloor \frac{n}{4} \rfloor$ points for each iteration.
- Each such iteration takes $\mathcal{O}(n)$ time.
- Therefore, the time complexity of the above algorithm for the constrained 1-center problem is:

$$\begin{aligned} T(n) &= T(\tfrac{3n}{4}) + \mathcal{O}(n) \\ &= \mathcal{O}(n) \end{aligned}$$