COMP 446 / 546 ALGORITHM DESIGN AND ANALYSIS

LECTURE 13 LINEAR PROGRAMMING

ALPTEKİN KÜPÇÜ

Based on slides of Xukai Zou, Shafi Goldwasser, Jeff Erickson, and Alexander Zelikovsky

LINEAR PROGRAMMING

- **A very general problem that can be used to express a wide variety of problems.**
  - e.g., max-flow and min-cut
  - Many more optimization problems…
- **Focus on how to express various problems as linear programs**
- **At the end, briefly discuss an algorithm for solving linear programming problems**

EXAMPLE SCENARIO

- **Politics: How to campaign to win an election**

Estimate votes obtained per dollar spent advertising on a particular issue:

Variable	Policy	urban	suburban	rural
			demographic	
x_1	building roads	-2	5	3
x_2	gun control	8	2	-5
x_3	farm subsidies	0	0	10
x_4	gasoline tax	10	0	-2

Want to win majority in each demographic by spending minimum amount of money

demographic:	urban	suburban	rural
population:	100.000	200.000	50.000
majority:	50.000	100.000	25.000

LINEAR PROGRAMMING DEFINITION

- **Suppose all the numbers below are real numbers.**
- **Given a linear function (called objective function)**
  - $f(x_1, x_2, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \ldots + c_n x_n = \sum_{j=1}^{n} c_j x_j$
- **With constraints**
  - $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ for $i=1,2,\ldots,m$ (linear constraints)
  - $x_j \geq 0$ for $j=1,2,\ldots,n$ (non-negativity)
- **Find values for $x_1, x_2, \ldots, x_n$ which maximize $f(x_1, x_2, \ldots, x_n)$**
- **Or change $\leq b_i$ to $\geq b_i$, then minimize $f(x_1, x_2, \ldots, x_n)$**

LINEAR PROGRAMMING

- **Vector Form**

Maximize: cx

Subject to: $Ax \leq b$

$c = (c_1, c_2, \ldots, c_n)$

$x = \begin{pmatrix} x_1 \\ . \\ . \\ x_n \end{pmatrix}$ $\quad b = \begin{pmatrix} b_1 \\ . \\ . \\ b_n \end{pmatrix}$

$A = \begin{pmatrix} a_{11} \ldots a_{1n} \\ \ldots\ldots \\ \ldots\ldots \\ a_{n1} \ldots a_{mn} \end{pmatrix}$

- **Summation Form**

Maximize: $\sum c_i x_i$

Subject to:

$\sum a_{1i} x_i \leq b_1$

$\sum a_{2i} x_i \leq b_2$

.

.

$\sum a_{mi} x_i \leq b_m$

EXAMPLE SCENARIO

Variable	Policy	urban	suburban	rural
			demographic	
x_1	building roads	-2	5	3
x_2	gun control	8	2	-5
x_3	farm subsidies	0	0	10
x_4	gasoline tax	10	0	-2
	Population:	100K	200K	50K
	Majority:	50K	100K	25K

- **4 variables**
  - x_1 is the # of thousands of dollars on advertising on building roads.
  - x_2 is the # of thousands of dollars on advertising on gun control.
  - x_3 is the # of thousands of dollars on advertising on farm subsidies.
  - x_4 is the # of thousands of dollars on advertising on gasoline tax.

EXAMPLE SCENARIO

Variable	Policy	urban	demographic suburban	rural
x_1	building roads	-2	5	3
x_2	gun control	8	2	-5
x_3	farm subsidies	0	0	10
x_4	gasoline tax	10	0	-2
	Population:	100K	200K	50K
	Majority:	50K	100K	25K

Formulate this problem as:

$$
\begin{array}{llllllll}
\text{Minimize:} & x_1 & + & x_2 & + & x_3 & + & x_4 & \\
\text{Subject to:} & -2x_1 & + & 8x_2 & + & 0x_3 & + & 10x_4 & \geq 50 \\
 & 5x_1 & + & 2x_2 & + & 0x_3 & + & 0x_4 & \geq 100 \\
 & 3x_1 & + & -5x_2 & + & 10x_3 & + & -2x_4 & \geq 25 \\
 & x_1, & x_2, & x_3, & x_4 & & & & \geq 0
\end{array}
$$

The solution of this linear program yields an optimal strategy for the politician (minimizes the cost while getting majority votes).

LINEAR PROGRAMMING IN STANDARD FORM

- **Standard form contains all non-negativity constraints, and only inequality type linear constraints**
  - Can represent any linear program in standard form.
- **If some variable x_i, does not have non-negativity constraint:**
  - Delete x_i but introduce two variables x_i' and x_i'',
  - Replace each occurrence of x_i with $x_i'-x_i''$.
  - Add constraints: $x_i' \geq 0$ and $x_i'' \geq 0$.

LINEAR PROGRAMMING IN STANDARD FORM

- **If there are equality constraints:**
  - Replace $\sum_{j=1}^{n} a_{ij}x_j = b_i$ with $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ and $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$
  - Then change $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$ to $\sum_{j=1}^{n} a_{ij}x_j \geq -b_i$ for minimization
  - Or $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ to $\sum_{j=1}^{n} a_{ij}x_j \leq -b_i$ for maximization.

LINEAR PROGRAMMING IN SLACK FORM

- **In slack form, except non-negativity constraints, all other constraints are equalities.**
- **Change standard form to slack form:**
  - If $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, then introduce new variable s, and set:
  - $s_i = b_i - \sum_{j=1}^{n} a_{ij}x_i$ and $s_i \geq 0$ (i=1,2,…,m).
  - If $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$, then introduce new variable s, and set:
  - $s_i = \sum_{j=1}^{n} (a_{ij}x_j) - b_i$ and $s_i \geq 0$ (i=1,2,…,m).

LP FOR SHORTEST PATH

- **(Single pair) Shortest path:**
  - Given a weighted directed graph G=<V,E> with weighted distance function w: $E \rightarrow R$, a source s and a target t
  - Compute $d[t]$ which is the total weighted distance of the shortest path from s to t.
- **Change to LP**
  - For each vertex v, introduce a variable $d[v]$: the distance of the shortest path from s to v.

maximize: $\quad \mathbf{d}[t]$

subject to: $\quad \mathbf{d}[v] \leq \mathbf{d}[u] + \mathbf{w}(u, v)$ **for each** $(u, v) \in E$

$\quad \mathbf{d}[s] = 0$

LP FOR SHORTEST PATH

maximize: d[t]

subject to: d[v] ≤ d[u] + w(u, v) for each (u, v) ∈ E

d[s] = 0

- ***What if there are negative weights?***
- **Add additional constraint: d[v] ≥ 0 for all v ∈ V.**
- **If the LP has no solution, it means the graph has a negative-weight cycle that can be reached from s.**
  - (now this LP is equivalent to the Bellman-Ford algorithm)

LP FOR MAXIMUM FLOW

- **Max-flow problem:**
  - Given a directed graph $G=\langle V,E\rangle$, a capacity function on each edge $c(u,v) \geq 0$, source s and target t.
  - A flow is a function $f : V \times V \rightarrow R$ that satisfies:
    - Capacity constraints: for all $(u,v) \in E$, $f(u,v) \leq c(u,v)$
    - Flow conservation: for all $u \in V\text{-}\{s,t\}$, $\sum_{(u,u)\in E} f(u,v) = \sum_{(u,v)\in E} f(u,v)$
  - Find a maximum flow from s to t.

maximize: $\sum f(s,v)$ **OR** $\sum f(v,t)$

subject to:

$f(u,v) \leq c(u,v)$	for each $(u,v) \in E$
$\sum_{(u,u)\in E} f(u,v) = \sum_{(u,v)\in E} f(u,v)$	for each $u \in V - \{s,t\}$
$0 \leq f(u,v)$	for each $(u,v) \in E$

DUALITY

Duality:

$$\max \vec{c} \cdot \vec{x} \qquad \equiv \qquad \min \vec{b}^{\mathsf{T}} \cdot \vec{y}$$

$$\text{s.t.} \quad A\vec{x} \le \vec{b} \qquad\qquad \text{s.t.} \quad A^{\mathsf{T}}\vec{y} \ge \vec{c}$$

$$\vec{x} \ge 0 \qquad\qquad \vec{y} \ge 0$$

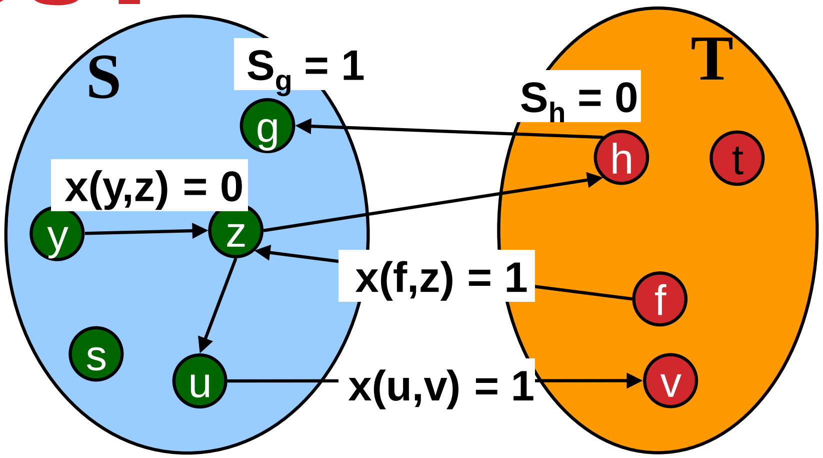
Example:

MAX-FLOW **MIN-CUT**

Important: if LP is unbounded (optimal value is $\pm\infty$) then dual LP is infeasible (has no solution)

If they have a solution, their values are equal.

LP FOR MINIMUM CUT



- **Minimum cut: (S, T = V-S)**
  - S_v : is $v \in S$? (0 or 1)
  - x(u,v) : is $u \in S$ & $v \in T$? (0 or 1)

minimize: $\sum_{(u,v)\in E} c(u,v) \cdot x(u,v)$

subject to:

	$S_u - S_v \leq x(u,v)$	**for each $(u,v) \in E$**
	$S_s = 1$	**(source is in S)**
	$S_t = 0$	**(target is not in S)**
	$x(u,v) \geq 0$	**for each $(u,v) \in E$**

- **We want to be able to say**
  - $S_u \in \{0,1\}$ for all $u \in V$
  - $x(u,v) \in \{0,1\}$ for all $(u,v) \in E$

These are not linear constraints !!

INTEGER LINEAR PROGRAMMING

Vector Form

Maximize: cx

Subject to : $Ax \leq b$

and $x \in \{0,1\}$

$c = (c_1, c_2, \ldots, c_n)$

$$x = \begin{pmatrix} x_1 \\ . \\ . \\ x_n \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ . \\ . \\ b_n \end{pmatrix}$$

$$A = \begin{pmatrix} a_{11} \ldots a_{1n} \\ \ldots\ldots \\ \ldots\ldots \\ a_{n1} \ldots a_{mn} \end{pmatrix}$$

Summation Form

Maximize: $\sum c_i x_i$

Subject to:

$\sum a_{1i} x_i \leq b_1$

$\sum a_{2i} x_i \leq b_2$

.

.

$\sum a_{mi} x_i \leq b_m$

and $x \in \{0,1\}$

In general, x must be an integer

Unfortunately, ILP is NP-complete!!

ILP FOR MAXIMUM INDEPENDENT SET

- **Maximum Independent Set (MIS)**
  - Given a graph $G = (V, E)$
  - Find the maximum subset of nodes which are pairwise non-adjacent (independent)
- **ILP: For any $v \in V$ make a variable $x_v \in \{0, 1\}$**
  - $x_v = 0 \Leftrightarrow v \notin$ MIS which means v is not chosen
  - $x_v = 1 \Leftrightarrow v \in$ MIS which means v is chosen

maximize:	$\sum_{v \in V} x_v$	
subject to:	$x_u + x_v \leq 1$	$\forall (u, v) \in E$
	$x_v \in \{0, 1\}$	$\forall v \in V$

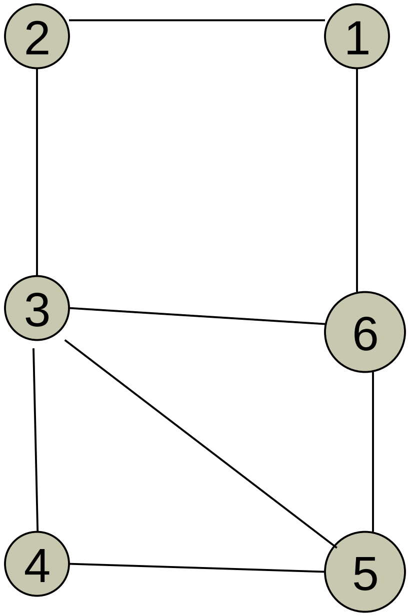
EXAMPLE ILP: MAXIMUM INDEPENDENT SET

maximize: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$

subject to:

$$x_1 + x_6 \leq 1$$
$$x_1 + x_2 \leq 1$$
$$x_2 + x_3 \leq 1$$
$$x_3 + x_6 \leq 1$$
$$x_3 + x_5 \leq 1$$
$$x_6 + x_5 \leq 1$$
$$x_3 + x_4 \leq 1$$
$$x_4 + x_5 \leq 1$$

and $x_1, x_2, \ldots, x_6 \in \{0,1\}$

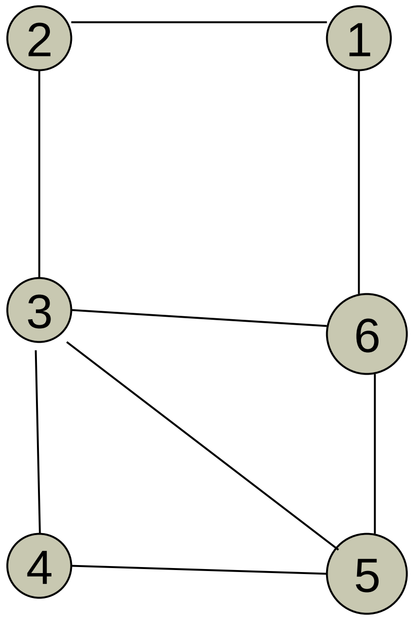


LP RELAXATION (LPR) VS. ILP

- **LP relaxation (LPR) for Max Independent Set problem (MIS) may provide a different solution than the actual maximum independent set.**
  - The difference is called integrality gap.
- **For some problems, a carefully-crafted LPR works really well…**

ILP for MIS	**LPR for MIS**
maximize: $\sum x_i \quad i \subseteq V$	**maximize:** $\sum x_i \quad i \subseteq V$
subject to:	**subject to:**
$x_i + x_j \leq 1$**, for each edge** $(x_i,x_j) \in E$	$x_i + x_j \leq 1$**, for each edge** $(x_i,x_j) \in E$
$x_i \in \{0, 1\}$	$0 \leq x_i \leq 1$

EXAMPLE LPR: MAXIMUM INDEPENDENT SET



maximize: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$

subject to:

$$x_1 + x_6 \le 1$$
$$x_1 + x_2 \le 1$$
$$x_2 + x_3 \le 1$$
$$x_3 + x_6 \le 1$$
$$x_3 + x_5 \le 1$$
$$x_6 + x_5 \le 1$$
$$x_3 + x_4 \le 1$$
$$x_4 + x_5 \le 1$$

and $0 \le x_1, x_2, \ldots, x_6 \le 1$

ILP

$x_1 = x_3 = x_5 = 1$

$\Sigma x_i = 3$

LPR

$x_i = ½$

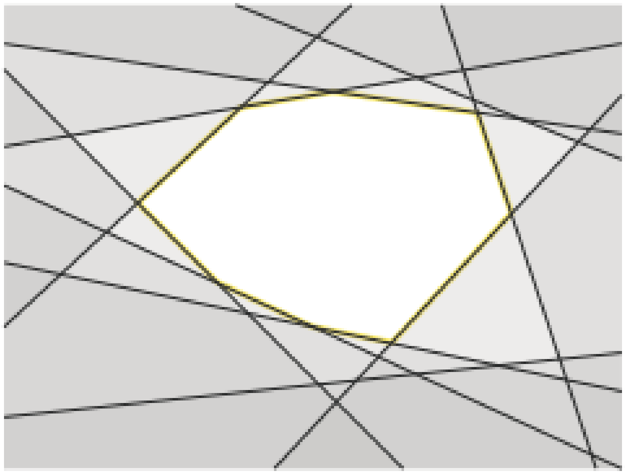
$\Sigma x_i = 3$

HOW TO SOLVE LP

- **OK, cool, now we know the definition an LP.**
- **YES, I can convert a given optimization problem to LP.**
  - Not easy for all problems though.
  - But I can do it for many problems.
- **If so, you are ready to learn about solving an LP rather than just formulating it…**
  - Hint: It might be easier to keep the matrix formulation in mind, and seeing n-tuples as a point in n-dimensional space.

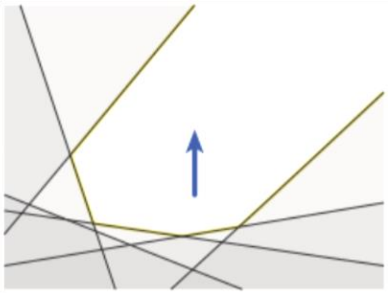
GEOMETRIC VIEW OF LP

- $\vec{x}$ is a point in R^d
- $\vec{c}$ is a direction vector (length is irrelevant)
- Maximize: $\vec{c} \cdot \vec{x}$ (find $\vec{x}$ most in the direction $\vec{c}$)
- Constraints: $\vec{a} \cdot \vec{x} \leq b$ (halfspace bounded by plane)
  - Constraints form a polytope with $\leq n$ polygonal facets
  - Possibly unbounded ("open")

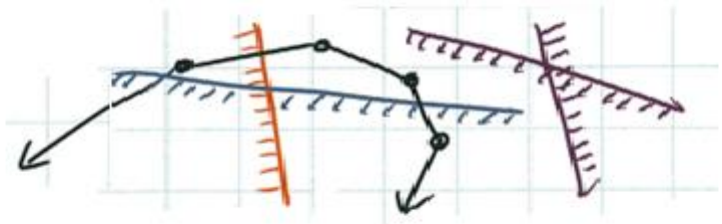


GEOMETRIC VIEW OF LP

- $\vec{x}$ is a point in R^d
- $\vec{c}$ is a direction vector (length is irrelevant)
- Maximize: $\vec{c} \cdot \vec{x}$ (find $\vec{x}$ most in the direction $\vec{c}$)
- Constraints: $\vec{a} \cdot \vec{x} \leq$ b (halfspace bounded by plane)
  - Constraints form a polytope with $\leq n$ polygonal facets
  - Possibly unbounded (“open”)
- Can rotate entire problem so that goal is to find highest point $\vec{x}$ in polytope
  - Essentially, $\vec{c} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$



GEOMETRIC VIEW OF LP



- **Incremental Algorithm for 2D**
  - Maintain polygon for first i-1 constraints
    - Must maintain polygon in balanced search tree
  - Add i^{th} constraint:
    - 0, 1, or 2 intersections
    - Can find in O(log n) similar to binary search
  - Discard now-irrelevant constraints
- **O(n log n) time**

- **Higher dimensions:**
  - # vertices = $\binom{n}{d} \approx n^d$ in the worst case (EXPONENTIAL)

ALGORITHMS FOR SOLVING LP

- **Simplex algoritm: $\vec{x}$ walks from vertex to vertex in direction $\vec{c}$**
  - practical but worst-case exponential (see book)
- **Ellipsoid algorithm: Guarantee OPT ∈ ellipsoid; reduce ellipsoids.**
  - First poly-time, useful in theory, impractical
- **Interior-point method: $\vec{x}$ flows inside polytope vaguely in direction of $\vec{c}$**
  - Poly-time & quite practical
- **Random sampling: [Bertsimas & Vempala 2004]**
  - Sample to estimate center of mass, slice OPT estimate, repeat
- **Randomized simplex: [Kelner & Spielman 2006]**
  - Reduce to testing boundedness; randomize $\vec{b}$; simplex; repeat

SIMPLEX ALGORITHM

- **Works well in practice**
  - Fast in general
  - Worst-case exponential
- **Algorithmically simple concept**
- **Demonstrate using an example**

SIMPLEX EXAMPLE

Maximize: $3x_1+x_2+2x_3$ (29.53)

Subject to:

$x_1+x_2+3x_3 \leq 30$ (29.54)

$2x_1+2x_2+5x_3 \leq 24$ (29.55)

$4x_1+x_2+2x_3 \leq 36$ (29.56)

$x_1, x_2, x_3 \geq 0$ (29.57)

Slack form:

Maximize: $z = 3x_1+x_2+2x_3$ (29.58)

Subject to:

$x_4 = 30-x_1-x_2-3x_3$ (29.59)

$x_5 = 24-2x_1-2x_2-5x_3$ (29.60)

$x_6 = 36-4x_1-x_2-2x_3$ (29.61)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

SIMPLEX DEFINITIONS

- **Feasible solutions:**
  - An assignment to variables whose values satisfy all constraints.
  - There are infinitely-many feasible solutions since we work with real numbers.
- **Basic solution:**
  - Basic variables: left-hand side variables
  - Free variables: right-hand side variables
  - Set all free variables to 0 and compute all basic variables to obtain basic solution
- **Basic feasible solution:**
  - When a basic solution is also feasible

SIMPLEX ALGORITHM STEPS

- **Iteratively re-write the set of equations such that**
  - No change to the underlying LP problem
  - The feasible solution set stays the same
- **However the basic solution changes, resulting in a greater objective value each time:**
  - Select a free variable x_i whose coefficient in objective function is positive
  - Increase value of x_i as much as possible without violating any of constraints
  - Re-write the problem such that x_i is now basic and some other variable is now free.

SIMPLEX EXAMPLE

Slack form:

Maximize: $z = 3x_1+x_2+2x_3$ (29.58)

Subject to:

$x_4 = 30-x_1-x_2-3x_3$ (29.59)

$x_5 = 24-2x_1-2x_2-5x_3$ (29.60)

$x_6 = 36-4x_1-x_2-2x_3$ (29.61)

$x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6 \geq 0$

Free variables: x_1,x_2,x_3

Basic variables: x_4,x_5,x_6

Basic solution:

$(x_1,x_2,x_3,x_4,x_5,x_6) = (0,0,0,30,24,36)$

$z = 3 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 = 0$

SIMPLEX EXAMPLE

Slack form:

Maximize: $z = 3x_1+x_2+2x_3$ (29.58)

Subject to:

$x_4 = 30-x_1-x_2-3x_3$ (29.59)

$x_5 = 24-2x_1-2x_2-5x_3$ (29.60)

$x_6 = 36-4x_1-x_2-2x_3$ (29.61)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Try to increase the value of x_1:

$x_4=30-x_1-x_2-3x_3$ $(x_1 \leq 30)$

$x_5=24-2x_1-2x_2-5x_3$ $(x_1 \leq 12)$

$x_6=36-4x_1-x_2-2x_3$ $(x_1 \leq 9)$

Increase only up to 9 (limiting equation is 29.61)

Write $x_1 = 9 – x_2/4 – x_3/2 – x_6/4$

Replace on all equations in the slack form

Now x_1 is basic and x_6 is free

SIMPLEX EXAMPLE

Slack form:

Maximize: $z = 3x_1+x_2+2x_3$ (29.58)

Subject to:

$x_4 = 30-x_1-x_2-3x_3$ (29.59)

$x_5 = 24-2x_1-2x_2-5x_3$ (29.60)

$x_6 = 36-4x_1-x_2-2x_3$ (29.61)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Updated Equivalent Slack form:

Maximize: $z = 27 + x_2/4 + x_3/2 - 3x_6/4$ (29.64)

Subject to:

$x_4 = 21 - 3x_2/4 - 5x_3/2 + x_6/4$ (29.66)

$x_5 = 6 - 3x_2/2 - 4x_3 + x_6/2$ (29.67)

$x_1 = 9 - x_2/4 - x_3/2 - x_6/4$ (29.65)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

SIMPLEX ALGORITHM PIVOTING

- **This operation is called pivoting**
  - A pivot operation chooses a free (entering) variable and a basic (leaving) variable, and changes their roles.
  - The pivot operation results in an equivalent LP.
    - It can be checked the original solution (0,0,0,30,24,36) satisfies the new equations.
- **In the example**
  - x_1 is entering variable, and x_6 is leaving variable.
  - x_2, x_3, x_6 are now free, and x_1, x_4, x_5 are now basic.
  - The basic solution for this is (9,0,0,21,6,0), with z = 27.
    - Remember, all free variables are set to 0 in the basic solution
    - Improved objective value!

SIMPLEX EXAMPLE

Updated Equivalent Slack form:

Maximize: $z = 27 + x_2/4 + x_3/2 - 3x_6/4$ (29.64)

Subject to:

$x_4 = 21 - 3x_2/4 - 5x_3/2 + x_6/4$ (29.66)

$x_5 = 6 - 3x_2/2 - 4x_3 + x_6/2$ (29.67)

$x_1 = 9 - x_2/4 - x_3/2 - x_6/4$ (29.65)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Try to increase the value of x_3: (x_2 also works but not x_6)

$x_4 = 21\text{-}3x_2/4 - 5x_3/2 + x_6/4$ ($x_3 \leq 42/5$)

$x_5 = 6\text{-}3x_2/2 - 4x_3 + x_6/2$ ($x_3 \leq 3/2$)

$x_1 = 9\text{-}x_2/4 - x_3/2 - x_6/4$ ($x_3 \leq 18$)

Increase only up to 3/2 (limiting equation is 29.67)

Write $x_3=3/2 - 3x_2/8 - x_5/4 + x_6/8$

Replace on all equations in the slack form

Now x_3 is basic and x_5 is free

SIMPLEX EXAMPLE

Updated Equivalent Slack form (2):

Maximize: $z = 111/4 + x_2/16 - x_5/8 - 11x_6/16$ (29.68)

Subject to:

$x_1 = 33/2 - x_2/16 + x_5/8 - 5x_6/16$ (29.69)

$x_3 = 3/2 - 3x_2/8 - x_5/4 + x_6/8$ (29.70)

$x_4 = 69/4 + 3x_2/16 + 5x_5/8 - x_6/16$ (29.71)

$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$

Free variables: x_2, x_5, x_6

Basic variables: x_1, x_3, x_4

Basic solution:

$(x_1, x_2, x_3, x_4, x_5, x_6) = (33/4, 0, 3/2, 69/4, 0, 0)$

$z = 111/4$

NEXT: Try to increase the value of x_2: ***ON THE BOARD***

SIMPLEX EXAMPLE

Updated Equivalent Slack form (3):

Maximize: $z = 28 - x_3/6 - x_5/6 - 2x_6/3$

Subject to:

$$x_1 = 8 + x_3/6 + x_5/6 - x_6/3$$
$$x_2 = 4 - 8x_3/3 - 2x_5/3 + x_6/3$$
$$x_4 = 18 - x_3/2 + x_5/2$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \geq 0$$

At this point, all coefficients in objective function are negative. So no further rewrite can be done. We have found the optimal solution.

Basic solution:

$$(x_1,x_2,x_3,x_4,x_5,x_6) = (8,4,0,18,0,0) \text{ with } z = 28$$

The original variables are x_1, x_2, x_3 , with values (8,4,0)
The original objective value of $z = 3x_1 + x_2 + 2x_3$ is 28

SIMPLEX RUNNING TIME

- **Lemma 29.7:**
  - Assuming that INITIALIZE-SIMPLEX returns a slack form for which the basic solution is feasible, SIMPLEX either reports that a linear program is unbounded, or it terminates with a feasible solution in at most $\binom{m+n}{m}$ iterations.
    - n+m variables
    - m equations
    - in the slack form

- **Note: Simplex may cycle indefinitely (if not implemented carefully).**
  - See Lemma 29.2 and Lemma 29.6

SIMPLEX QUESTIONS

- ***How do we determine whether a linear program is unbounded?***
  - When no equation bounds a variable, then the LP is unbounded
  - Simplex algorithm detects this situation
- ***How do we choose the entering and leaving variables?***
  - Bland's rule: Pick the variable with lowest index that has positive coefficient in the objective function
    - Prevents indefinite cycling
- ***How do we determine whether a linear program is feasible?***
- ***What do we do if the linear program is feasible, but the initial basic solution is not feasible?***

AUXILIARY LP

- **Lemma 29.11:**
  - Let *L* be a linear program in standard form.
  - Let x_0 be a new variable, and let L_{aux} be the following linear program with n+1 variables

maximize: $$-x_0 \qquad (29.106)$$

subject to:

$$\sum_{j=1}^{n}(a_{ij}x_j) - x_0 \le b_i \qquad \text{for } i = 1, 2, \ldots, m \qquad (29.107)$$

$$x_j \ge 0 \qquad \text{for } j = 0, 1, \ldots, n \qquad (29.108)$$

- **Then *L* is feasible if and only if the optimal objective value of L_{aux} is 0.**

PROOF ??

FUNDAMENTAL THEOREM OF LINEAR PROGRAMMING

- **Theorem 29.13:**
  - Any linear program L, given in standard form, either
    - 1. has an optimal solution with a finite objective value, or
    - 2. is infeasible, or
    - 3. is unbounded.
- **If L is infeasible, SIMPLEX returns "infeasible". If L is unbounded, SIMPLEX returns "unbounded". Otherwise, SIMPLEX returns an optimal solution with a finite objective value.**

http://en.wikipedia.org/wiki/Linear_programming#Open_problems_and_recent_work

http://en.wikipedia.org/wiki/Time_complexity#Strongly_and_weakly_polynomial_time

CONCLUSIONS

- **Linear Programming is a very useful and powerful way of formulating optimization problems**
- **Simplex is a practically-efficient algorithm to solve LP problems**
  - Other algorithms with better theoretical guarantees are known
  - The practical performance of Simplex, despite its theoretical exponential worst-case, has challenged many researchers, and much more detailed analysis on Simplex have been performed
  - This has led to new areas on analysis of algorithms
  - An important problem is finding an input that gives the worst-case performance for a given algorithm
    - e.g., reverse-sorted array for quick sort (with the first element as pivot)
    - for Simplex, an input with run-time 2^n-1 iterations is known