Linear Programming

- Another "Sledgehammer" in our toolkit
- Many problems fit into the Linear Programming approach
- These are optimization tasks where both the constraints and the objective are linear functions
- Given a set of variables we want to assign real values to them such that they
  - Satisfy a set of variable constraints represented by linear equations and/or linear inequalities
  - Maximize/minimize a given linear objective function
- We will consider a few examples
  - We want to get good at a) recognizing a problem which can be represented with LP and b) casting it into an LP formulation
The Chocolate Shop

You own a chocolate shop which produces two types of box chocolates:
- Normal box which gives a $1 profit
- Deluxe box which gives a $6 profit

The variables are the number of boxes produced per day
- $x_1$ is the number of boxes of normal chocolate
- $x_2$ is the number of boxes of deluxe chocolate

The objective is to set $x_1$ and $x_2$ so to maximize profit
- $\max (x_1 + 6x_2)$  \hspace{1cm} \text{Profit} = x_1 + 6x_2$

The constraints are:
- $x_1 \leq 200$  \hspace{1cm} \text{Maximum demand of normal boxes per day}
- $x_2 \leq 300$  \hspace{1cm} \text{Maximum demand of deluxe boxes per day}
- $x_1 + x_2 \leq 400$  \hspace{1cm} \text{Maximum production capacity}
- $x_1, x_2 \geq 0$  \hspace{1cm} \text{Can't have a negative number of boxes}
Representing a Linear Program

- The objectives and constraints define a linear program.
- Easy to visualize in low dimensions (2 or 3 variables)
  - Feasible space forms a convex polygon
- Optimum is achieved at a vertex, except when
  - No solution to the constraints
  - Feasible region is unbounded in direction of the objective

(a) \[
\begin{array}{c}
\text{x}_2 \\
\hline
400 \\
300 \\
200 \\
100 \\
0 \\
\end{array}
\]
\[x_1\]

(b) \[
\begin{array}{c}
\text{x}_2 \\
\hline
400 \\
300 \\
200 \\
100 \\
0 \\
\end{array}
\]
\[x_1\]

Optimum point
Profit = $1900

- \(c = 1500\)
- \(c = 1200\)
- \(c = 600\)
Solving a Linear Program

- Simplex Algorithm Preview – More details to follow
  - Start at any vertex
  - If a neighbor has a better objective value, move to it, otherwise already at optimum

- Is this optimal?
  - Convexity and linearity the reason

- If more than one neighbor with a better objective value, which one should you go to?
The Chocolate Shop - Expanded

- You add a third product: Supreme box with $13 profit and a packaging constraint that $x_2 + 3x_3 \leq 600$
- As dimensions increase beyond 3, no longer visualizable, but LP still works and simplex still solves it

![Graph showing a linear programming problem with constraints and objective function: max $x_1 + 6x_2 + 13x_3$ with $x_1 \leq 200$, $x_2 \leq 300$, $x_1 + x_2 + x_3 \leq 400$, $x_2 + 3x_3 \leq 600$, $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$]
Divisible Knapsack with Constraints

Divisible knapsack allows us to put fractional items in our bag

Set up Linear Program assuming
- 2 items of weights 6 and 2 with values $10 and $4 respectively
- \( W = 9 \)
- We want no more than 3 items in the bag (not part of standard knapsack)
- Let's set it up and solve it (would if weights had been 5 and 2)

Can add arbitrary constraints quite easily
- exactly \( n \) of item \( m \), no more than \( z \) of items \( a \) and \( b \) combined, etc.

0-1 knapsack – would require us to constrain variables to be integers
- Integer Linear Programming
- ILP is harder than standard LP – no known polynomial solution
Reductions

- Sometimes a task is sufficiently general that the algorithm for it can be used to solve many other tasks, even ones that initially seem unrelated.

- For example, if we have an algorithm \( \text{shortestpath}(G) \) then we could use it to solve the problem of finding the longest path as follows:

  \[
  \text{longestpath}(G) \\
  \text{negate all edge weights in } G \\
  \text{return } \text{shortestpath}(G)
  \]

- We say that \( \text{longestpath} \) reduces to \( \text{shortestpath} \).
Note that we had to do some preprocessing/postprocessing of the data before we could run it on shortestpath($G$).

As long as $O(\text{pre/post process}) \leq O(Q)$, then $O(P) = O(Q)$.

In this class you are learning a number of classic algorithms to which many other tasks can reduce.

In future you will say, "Oh, that problem is just about like knapsack, or Dijkstra, etc."

Then you can either create slightly modified algorithms or reduce directly to algorithms you understand well in terms of both execution and complexity.
In general an LP could
- be either a maximization or a minimization problem
- include both equalities and inequalities in the constraints
- have negative and positive variables

By a reduction we can take LPs of any of these forms and cast them into one equivalent *standard form*
- always a minimization/maximization problem
- all constraints are either equalities or inequalities
- only has positive variables

Since $O(\text{reduction}) < O(\text{LP})$, do not lose time complexity

This could be convenient for allowing simpler LP solving software or for reasoning about LP problems, etc.
A maximization problem can be turned into a minimization problem by negating the coefficients of the objective function.

Turn an inequality into an equality by introducing a *slack variable* $s$ as follows:

\[ \sum a_i x_i \leq b \quad \text{becomes} \quad \sum a_i x_i + s = b \quad \text{and} \quad s \geq 0 \]

Could have turned equalities into inequalities:

\[ ax = b \quad \text{becomes} \quad ax \leq b \quad \text{and} \quad ax \geq b \]

Replace an arbitrary $x$ (positive or negative) with $x^+ - x^-$ where $x^+$ and $x^-$ are positive variables.

Note that this standard form comes at the price of adding variables and/or constraints, which will increase solution time.

| max $x_1 + 6x_2$ | \[ \begin{align*} x_1 \leq 200 \\
| x_2 \leq 300 \\
| x_1 + x_2 \leq 400 \\
| x_1, x_2 \geq 0 \end{align*} \right| \Rightarrow \begin{align*} \min -x_1 - 6x_2 \\
| x_1 + s_1 = 200 \\
| x_2 + s_2 = 300 \\
| x_1 + x_2 + s_3 = 400 \\
| x_1, x_2, s_1, s_2, s_3 \geq 0 \end{align*} \] |
Matrix-vector notation

A linear function like $x_1 + 6x_2$ can be written as the dot product of two vectors

$$c = \begin{pmatrix} 1 \\ 6 \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

denoted $c \cdot x$ or $c^T x$. Similarly, linear constraints can be compiled into matrix-vector form:

$$\begin{align*}
x_1 & \leq 200 \\
x_2 & \leq 300 \\
x_1 + x_2 & \leq 400
\end{align*} \quad \Rightarrow \quad \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 200 \\ 300 \\ 400 \end{pmatrix}.$$

Here each row of matrix $A$ corresponds to one constraint: its dot product with $x$ is at most the value in the corresponding row of $b$. In other words, if the rows of $A$ are the vectors $a_1, \ldots, a_m$, then the statement $Ax \leq b$ is equivalent to

$$a_i \cdot x \leq b_i \quad \text{for all} \quad i = 1, \ldots, m.$$

With these notational conveniences, a generic LP can be expressed simply as

$$\begin{align*}
\max \quad & c^T x \\
A x & \leq b \\
x & \geq 0.
\end{align*}$$
Simplex Algorithm

- Basic form of the algorithm is very simple
  let $\nu$ be any vertex of the feasible region
  while there is a neighbor $\nu'$ of $\nu$ with better objective value:
    set $\nu = \nu'$
  return objective value of final $\nu$

- Always works because feasible region is a convex polyhedron and the objective function is linear

- Easy to visualize in low dimensions, but how do we actually implement it in high dimensional space
  - Must find the coordinates of each vertex
  - Must determine which vertices are neighbors
Simplex Algorithm – Basic Intuition

1. Put the LP in standard form (all constraints are equalities, variables $\geq 0$, and objective to be maximized)
2. Make initial vertex the origin
   - If origin is not a feasible vertex, the LP can be transformed to make it so, but we will assume that it is
3. At the origin all $x_i$ of the objective function are 0, so increasing any $x_i$ corresponding to a positive coefficient in the objective function will improve the objective value
   - If all objective coefficients are $\leq 0$, then current vertex is an optimum
4. Increase a chosen $x_i$ as far as possible (until some constraint is met/all the slack is taken out at the moment) and this will be a neighbor vertex and it will have a better objective value
5. Pivot the feasible polygon so that the last chosen vertex becomes the origin (change of basis) and adjust all LP equations accordingly. This causes the current chosen objective coefficient to become non-positive. Go to 3.
Simplex Example into Standard Form

\[
\begin{align*}
\text{maximize} & \quad 3x_1 + x_2 + 2x_3 \\
\text{subject to} & \quad x_1 + x_2 + 3x_3 \leq 30 \\
& \quad 2x_1 + 2x_2 + 5x_3 \leq 24 \\
& \quad 4x_1 + x_2 + 2x_3 \leq 36 \\
& \quad x_1, x_2, x_3 \geq 0
\end{align*}
\]
Simplex Example into Standard Form

Add slack variables to make all constraints equalities

Slack variables \((x_4 - x_6)\) are initially called *basic* variables and original variables \((x_1 - x_3)\) are called *non-basic* variables

Basic variables sit alone on the left-hand side of the constraint equations

Slack variables represent how far away a constraint is from being *tight*, which is a non-basic variable setting that makes the basic variable 0 (no more slack, and also finds neighbor vertex)

- \(x_4\) represents the difference between 30 and the sum of the non-basic variables times their coefficients – the available slack
Simplex Initial Values

\[
\begin{align*}
\text{maximize} & \quad 3x_1 + x_2 + 2x_3 \\
\text{subject to} & \quad x_1 + x_2 + 3x_3 \leq 30 \\
& \quad 2x_1 + 2x_2 + 5x_3 \leq 24 \\
& \quad 4x_1 + x_2 + 2x_3 \leq 36 \\
& \quad x_1, x_2, x_3 \geq 0
\end{align*}
\]

Maximize \( c^T x \) 
subject to \( Ax \leq b \) 
\( x \geq 0 \)

\[
\begin{align*}
z &= 3x_1 + x_2 + 2x_3 \\
x_4 &= 30 - x_1 - x_2 - 3x_3 \\
x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
x_6 &= 36 - 4x_1 - x_2 - 2x_3
\end{align*}
\]

\( x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \)

- Note that the signs in \( A \) match the original coefficients since \( Ax \leq b \) became \( s = b - Ax \)
- \( v \) is the current basic solution of the objective function
- \( B \) and \( N \) are indexes into the basic and non-basic variables
A basic solution is the solution when we set all non-basic variables (those on the right hand side) to 0 (the origin). Initially, the non-basic variables are $x_1$, $x_2$, and $x_3$, so $x = (0, 0, 0, 30, 24, 36)$ and the objective value $z = (3\cdot0) + (1\cdot0) + (2\cdot0) = 0$

We now need to move to a neighbor vertex with improved objective value.

To do this we choose any non-basic variable $x_e$ which has a positive coefficient in the objective function and increase it until it reaches a constraint, which will be a vertex with improved objective value.

Let's choose $x_1$. What value will it be given? Big as possible without violating any constraints.

- Set other non-basic variables to 0 since only moving on one dimension.
- If constraint equation does not contain $x_e$ or if the coefficient tied to $x_e$ in the equation is positive, then that basic variable will not constrain $x_e$.
- Otherwise, Solve for tightest constraint by setting basic (slack) variable to 0 since we want to find the value of $x_e$ using up all the slack.
- The $x_6$ constraint is tightest (hits 0 for smallest $x_1$ and thus has the least slack for $x_1$).
- Note that with $x_1$ increased, other basic variables will be decreased, but only $x_6$ goes all the way to 0.
- What is the new objective value?
The Pivot

- We now want to make the new vertex the origin so that we can do this all again.
- Note we increased the chosen non-basic variable \( x_e \) (entering) until a basic variable \( x_l \) (leaving) became 0
  - entering or leaving the column on the left hand side
- We will now make \( x_e \) (\( x_1 \)) a basic variable, and \( x_l \) (\( x_6 \)) will become a non-basic variable. This transformation of roles is the Pivot. The origin will then be translated and we once again will have non-basic variables that are all 0. (\( x_e \) and \( x_l \) swap/pivot roles)
  - The non-basic variables always make up the current objective function
- To do this we must adjust all the other basic variables by solving for \( x_e \) in terms of the non-basic variables including \( x_l \)
  - This is like your high school math assignment to solve \( n \) equations with \( n \) unknowns when using substitution
  - Also like the pivot in the Gauss elimination algorithm
Note that each basic variable is always defined in terms of $n$ non-basic variables.

\[
\begin{align*}
z &= 3x_1 + x_2 + 2x_3 \\
x_4 &= 30 - x_1 - x_2 - 3x_3 \\
x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
x_6 &= 36 - 4x_1 - x_2 - 2x_3
\end{align*}
\]

much we can increase: much we can increase. 

third constraint for 

\[
\begin{align*}
x_6 &= 36 - 4x_1 - x_2 - 2x_3 \\
4x_1 &= 36 - x_2 - 2x_3 - x_6 \\
x_1 &= 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}
\end{align*}
\]

We now rewrite the other equations by writing $x_1$ in terms of $x_2$, $x_3$, and $x_6$ using the above equation. Doing this for $x_4$ gives us

\[
\begin{align*}
x_4 &= 30 - x_1 - x_2 - 3x_3 \\
&= 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}\right) - x_2 - 3x_3 \\
&= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4}
\end{align*}
\]

We repeat this procedure for the remaining constraint and objective function to rewrite our linear program in the following form:

\[
\begin{align*}
z &= 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\
x_1 &= 9 - \frac{x_2}{4} - \frac{3x_3}{2} - \frac{x_6}{4} \\
x_4 &= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \\
x_5 &= 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2}
\end{align*}
\]

The rewrite operation as shown above is called a pivot. A pivot takes the non-basic variable $x_e$ called the entering variable and the basic variable $x_l$ called the leaving variable and exchanges their roles in the linear program.
The Next Neighbor

Now we can do the next basic solution by setting the current non-basic variables to 0: \( x = (9, 0, 0, 21, 6, 0) \)
- It is the new origin with \( x_2 \), \( x_3 \) and \( x_6 \) all set to 0

Objective function \( z \) above is 27 for this basic solution

Note that for this variable setting our original objective function must also be 27

What is the next non-basic variable to increase?
- \( x_2 \) or \( x_3 \) but not \( x_6 \) since it is negative and would lessen objective
- Let's do \( x_3 \)
Simplex Example Finished

\[
\begin{align*}
z &= \frac{111}{4} + \frac{x_2}{32} - \frac{x_5}{8} - \frac{11x_6}{16} \\
x_1 &= \frac{4}{3} - \frac{x_2}{16} + \frac{x_5}{8} - \frac{5x_6}{16} \\
x_3 &= \frac{3}{2} - \frac{3x_2}{8} - \frac{x_5}{4} + \frac{x_6}{8} \\
x_4 &= \frac{69}{4} + \frac{3x_2}{16} + \frac{5x_5}{8} - \frac{x_6}{16}
\end{align*}
\]

There is still a non-basic variable in the objective function with a positive coefficient. We increase this variable and pivot again to rewrite the linear program as:

\[
\begin{align*}
z &= 28 - \frac{x_3}{6} - \frac{x_5}{6} - \frac{2x_6}{3} \\
x_1 &= 8 + \frac{x_3}{6} + \frac{x_5}{6} - \frac{x_6}{3} \\
x_2 &= 4 - \frac{8x_3}{3} - \frac{2x_5}{3} + \frac{x_6}{3} \\
x_4 &= 18 - \frac{x_3}{2} + \frac{x_5}{2}
\end{align*}
\]

There are no other variables we can change to increase the objective value. The basic solution to the linear program is \((8, 4, 0, 18, 0, 0)\). The objective value from this solution is 28. We can now return to our original linear programming.

- Increasing \(x_3\) will cause a pivot with \(x_5\).
- Then only \(x_2\) has a positive coefficient and it will be increased and pivot with \(x_3\) and then we are done
- Final solution is \(x_1 = 8, x_2 = 4, x_3 = 0\), which gives \(z = 28\)
- Will 3 variables always require only 3 pivots? – No, path of vertices
Revisit Divisible Knapsack

- Let's set it up and walk through it with pictures of the feasible space
- Divisible knapsack allows us to put fractional items in our bag
- Set up Linear Program assuming
  - 2 items of weights 6 and 2 with values $10 and $4 respectively
  - $W = 9$
  - We want no more than 3 items in the bag
- Put it in standard matrix form
Knapsack Simplex Example

\[ z = 0 + 10x_1 + 4x_2 \]
\[ x_3 = 3 - x_1 - x_2 \quad (x_2 = -x_1 + 3) \]
\[ x_4 = 9 - 6x_1 - 2x_2 \quad (x_2 = -3x_1 + 4.5) \]

- Note that even though we now have 4 variables, we can still always just visualize the problem in terms of the non-basic variables (those in the current objective function).
- Select a variable to increase objective: \( x_1 \) or \( x_2 \)
- Let's do \( x_1 \) (note simplex does not see picture)
Knapsack Simplex Example

\[ z = 0 + 10x_1 + 4x_2 \]
\[ x_3 = 3 - x_1 - x_2 \quad (x_2 = -x_1 + 3) \]
\[ x_4 = 9 - 6x_1 - 2x_2 \quad (x_2 = -3x_1 + 4.5) \]

- Note that even though we now have 4 variables, we can still always just visualize the problem in terms of the non-basic variables (those in the current objective function).
- Select a variable to increase objective: \(x_1\) or \(x_2\).
- Tightest slack variable \(x_4\) gives \(x_1 = 1.5\).
  - \(x_3\) has also lost some slack (down to 1.5).
- Objective is now 15.
- We now need to move \((1.5,0)\) to the origin (The Pivot).
Knapsack Simplex Example

\[ z = 0 + 10x_1 + 4x_2 \]
\[ x_3 = 3 - x_1 - x_2 \quad (x_2 = -x_1 + 3) \]
\[ x_4 = 9 - 6x_1 - 2x_2 \quad (x_2 = -3x_1 + 4.5) \]

Note that even though we now have 4 variables, we can still always just visualize the problem in terms of the non-basic variables (those in the current objective function).

Select a variable to increase objective: \( x_1 \) or \( x_2 \).

Tightest slack variable \( x_4 \) gives \( x_1 = 1.5 \)

\( x_3 \) has also lost some slack (down to 1.5).

Objective is now 15.

We now need to move \((0,3)\) to the origin.

\[ x_1 = \frac{9}{6} - 2x_2/6 - x_4/6 = 1.5 - x_2/3 - x_4/6 \]
\[ z = 15 + 2x_2/3 - 10x_4/6 \]
\[ x_1 = 1.5 - x_2/3 - x_4/6 \]
\[ x_3 = 1.5 - 2x_2/3 + x_4/6 \]
Knapsack Simplex Example

\[
\begin{align*}
    z &= 0 + 10x_1 + 4x_2 \\
    x_3 &= 3 - x_1 - x_2 \quad (x_2 = -x_1 + 3) \\
    x_4 &= 9 - 6x_1 - 2x_2 \quad (x_2 = -3x_1 + 4.5)
\end{align*}
\]

Note that even though we now have 4 variables, we can still always just visualize the problem in terms of the non-basic variables (those in the current objective function).

Select a variable to increase objective: \(x_1\) or \(x_2\)

Tightest slack variable \(x_4\) gives \(x_1 = 1.5\)

- \(x_3\) has also lost some slack (down to 1.5)

Objective is now 15

We now need to move \((0,3)\) to the origin

\[
\begin{align*}
    x_1 &= 9/6 - 2x_2/6 - x_4/6 = 1.5 - x_2/3 - x_4/6 \\
    z &= 15 + 2x_2/3 - 10x_4/6 \\
    x_1 &= 1.5 - x_2/3 - x_4/6 \quad (x_2 = -x_4/2 + 4.5) \\
    x_3 &= 1.5 - 2x_2/3 + x_4/6 \quad (x_2 = x_4/4 + 2.25)
\end{align*}
\]

Which variable next? And how far?
Knapsack Simplex Example

- Move $x_2$ to 2.25
  - $x_2 = 2.25 - 3x_3/2 + x_4/4$
  - $z = 16.5 - x_3 - 3x_4/2$
  - $x_1 = .75 - x_3/2 - 3x_4$ ($x_3 = -6x_4 + 1.5$)
  - $x_2 = 2.25 - 3x_3/2 + 9x_4$ ($x_3 = 6x_4 + 1.5$)

- And the basic solution to this is our final answer
  - $x_1 = .75$ and $x_2 = 2.25$
  - Objective value is 16.5

Original problem:
  - $z = 0 + 10x_1 + 4x_2$ ($x_2 = -10x_1/4 + c$)
  - $x_3 = 3 - x_1 - x_2$ ($x_2 = -x_1 + 3$)
  - $x_4 = 9 - 6x_1 - 2x_2$ ($x_2 = -3x_1 + 4.5$)

- Would if slope of $z$ line had been $-1/4$?
// Previously convert the problem to standard form.
// We assume that the first basic solution is at the origin and is feasible.
// This is not true in general but makes our problem easier.
Simplex \((A, b, c)\)
\((N, B, A, b, c, v) = \text{InitializeSimplex}(A, b, c)\) //Initialize data structures
while there exists some \(j \in N\) such that \(c_j > 0\)
do choose an index \(e \in N\) such that \(c_e > 0\) // \(e\) is entering variable
    for each index \(i \in B\)
        do if \(a_{i,e} > 0\)
            then \(\delta_i = b_i/a_{i,e}\)
            else \(\delta_i = \infty\)
    choose \(l \in B\) such that \(\delta_i\) is minimized // \(l\) is leaving variable
    if \(\delta_l = \infty\)
        then return “unbounded”
    else \((N, B, A, b, c, v) = \text{Pivot} (N, B, A, b, c, v, l, e)\)
// set the non-basic variables to 0 and everything else to the optimal solution
for \(i = 1\) to \(n\)
    do if \(i \in B\)
        then \(\bar{x}_i = b_i\)
    else \(\bar{x}_i = 0\)
return \((\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)\)
Simplex Notes

- InitializeSimplex is just putting the variables in their data structures including the padded matrix $A$
  - You can enter the data in standard form. Your program can assume the data comes in standard form.
- Instead of $A$ being just the basic $m$ (constraints/basic variables) $\times$ $n$ (non-basic variables) matrix, you will pad it with 0's to be an $(n + m) \times (n + m)$ matrix with the basic $A$ matrix in the bottom left corner
- That way you have an easy place to store coefficients for all of the variables as you swap basic and non-basic variables
- $N$ and $B$ index the non-basic and basic variables
Padded Matrix Form: Initial Setup for Simplex

\[
\begin{align*}
\mathbf{z} &= 3x_1 + x_2 + 2x_3 \\
\mathbf{x}_4 &= 30 - x_1 - x_2 - 3x_3 \\
\mathbf{x}_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
\mathbf{x}_6 &= 36 - 4x_1 - x_2 - 2x_3 \\
\end{align*}
\]

\[
\begin{align*}
\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6 &\geq 0 \\
\mathbf{b}_T &= 0 0 0 30 24 36 \\
\mathbf{c}_T &= 3 1 2 0 0 0 \\
\mathbf{v} &= 0
\end{align*}
\]

\[
\begin{align*}
\mathbf{N} &= 1 2 3 \\
\mathbf{B} &= 4 5 6 \\
\mathbf{A} &= 0 0 0 0 0 0 0 0 0 \\
& 0 0 0 0 0 0 0 0 0 \\
& 0 0 0 0 0 0 0 0 0 \\
& 1 1 3 0 0 0 0 0 0 \\
& 2 2 5 0 0 0 0 0 0 \\
& 4 1 2 0 0 0 0 0 0 \\
\mathbf{b}_T &= 0 0 0 30 24 36 \\
\mathbf{c}_T &= 3 1 2 0 0 0 \\
\mathbf{v} &= 0
\end{align*}
\]

Remember \( \mathbf{Ax} = \mathbf{b} \), thus signs of equations above opposite of this \( \mathbf{A} \)
First minimum slack will be: \( b_6/a_{6,1} = 36/4 = 9 \) making \( e = 1 \) and \( l = 6 \)
After First Pivot

\[
\begin{align*}
z &= 3x_1 + x_2 + 2x_3 \\
x_4 &= 30 - x_1 - x_2 - 3x_3 \\
x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
x_6 &= 36 - 4x_1 - x_2 - 2x_3 \\
\end{align*}
\]

\[x_1, x_2, x_3, x_4, x_5, x_6 \geq 0\]

\[
\begin{align*}
z &= 27 + \frac{x_2}{4} + \frac{x_3}{2} - \frac{3x_6}{4} \\
x_1 &= 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \\
x_4 &= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \\
x_5 &= 6 - \frac{3x_2}{2} - 4x_3 + \frac{x_6}{2} \\
\end{align*}
\]

\[
\begin{align*}
N &= 6 & 2 & 3 \\
B &= 4 & 5 & 1 \\
A &= 0 & 1/4 & 1/2 & 0 & 0 & 1/4 \\
& & 0 & 0 & 0 & 0 & 0 \\
& & 0 & 0 & 0 & 0 & 0 \\
& & 0 & 3/4 & 5/2 & 0 & 0 & -1/4 \\
& & 0 & 3/2 & 4 & 0 & 0 & -1/2 \\
& & 0 & 0 & 0 & 0 & 0 & 0 \\
b^T &= 9 & 0 & 0 & 21 & 6 & 0 \\
c^T &= 0 & 1/4 & 1/2 & 0 & 0 & -3/4 \\
v &= 27 \\
\end{align*}
\]

First minimum slack was: \(b_6/a_{6,1} = 36/4 = 9\) making \(e = 1\) and \(l = 6\)
Pivot Algorithm

1. Pivot \((N, B, A, b, c, v, l, e)\)
2. \(/\ /\ \text{Compute coefficients for equation for the new basic variable } x_e\)
3. \(/\ /\ \text{The variables with hats, like } \hat{b}_e, \text{ will be the return values}\)
4. \(\hat{b}_e = b_l/a_{l,e}\)
5. \(\text{for each } j \in N - \{e\}\)
6. \(\quad \hat{a}_{e,j} = a_{l,j}/a_{l,e}\)
7. \(\hat{a}_{e,l} = 1/a_{l,e}\)
8. \(/\ /\ \text{Compute coefficients for the other constraints}\)
9. \(\text{for each } i \in B - \{l\}\)
10. \(\quad \hat{b}_i = b_i - a_{i,e}\hat{b}_e\)
11. \(\quad \text{for each } j \in N - \{e\}\)
12. \(\quad \hat{a}_{i,j} = a_{i,j} - a_{i,e}\hat{a}_{e,j}\)
13. \(\hat{a}_{i,l} = -a_{i,e}\hat{a}_{e,l}\)
14. \(/\ /\ \text{Compute the objective function}\)
15. \(\hat{v} = v + c_e\hat{b}_e\)
16. \(\text{for each } j \in N - \{e\}\)
17. \(\quad \hat{c}_j = c_j - c_e\hat{a}_{e,j}\)
18. \(\hat{c}_l = -c_e\hat{a}_{e,l}\)
19. \(/\ /\ \text{Compute new basic and non-basic variable sets}\)
20. \(\hat{N} = (N - \{e\}) \cup \{l\}\)
21. \(\hat{B} = (B - \{l\}) \cup \{e\}\)
22. return \((\hat{N}, \hat{B}, \hat{A}, \hat{b}, \hat{c}, \hat{v})\)
Simplex Complexity

- Each Simplex rewrite in the Pivot function has time complexity $O(mn)$, where $m$ is the # of constraints, and $n$ is the original # of variables.
- In Simplex we do not test all neighbors, but just select any one with a positive coefficient in the objective function.
- Thus total complexity is $O(m \cdot n \cdot \text{(number of vertices along the path to the optimum)})$.
- Upper bound on the total # of vertices is $\binom{m + n}{n}$ since each constraint is made up of $n$ variables and there are $m+n$ constraints.
- This number of possible total vertices is exponential and thus Simplex is an exponential algorithm.
- However, exponential examples do not occur in practice and thus Simplex is very powerful and widely used.
- There have recently been polynomial algorithms proven for LP (ellipsoid method and interior-points method), but empirically Simplex is still often more efficient.
Simplex Homework

- One careful walk through of the Simplex algorithm
- Need to understand the algorithm as you go
  - Can refer to the code if you wish
- Solve a different LP using an LP solver off the web
- One good approach for testing the web LP solver is to load in the linear program used in the handout/slides to see if you get the same results step by step
When to use Linear Programming

- Whenever we want to maximize some linear objective given a set of linear constraints
- Very common in many application areas
  - Business
  - Economics
  - Manufacturing
  - Design
  - Communications
  - Transportation
  - Planning
  - Diet
  - etc.
- Many software packages available
Non-Linear Programming

- And would if either objectives, constraints, or both are non-linear

Families of optimization algorithms
- Quadratic Optimization
- Software packages
- Takes into account just what kind of optimization it is
- Some get optimal, some approximate, depends on complexity of the objective and constraint functions

Local algorithms
- Popular algorithms which deal with arbitrarily complex problems with consistent relatively simple approaches
  - Does not require us to figure out first what are all the constraint complexities, etc.