12. Linear Programming Formulations

CPSC 535

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY
FULLERTON

Linear Programming Formulations

Many problems reduce to standard-form linear programming (LP):

- generalize standard form to be more flexible
  - allow minimization, negative variables, $=$ constraints, $\geq$ constraints
- business/administrative problems ("operations research")
- problems that don't resemble LP: shortest paths, max flow
- (similar situation to max-flow formulations)

LP Formulation Algorithms

```
1: function SOLVE-PROBLEM-A(input-to-A)
2:     LP = map input-to-A to a linear program
3:     outcome = SIMPLEX-LP-SOLVER(LP)
4:     if outcome is "unbounded" then
5:         return output in unbounded case
6:     else if outcome is "infeasible" then
7:         return output in infeasible case
8:     else
9:         output = map LP solution to A solution
10:        return output
11:    end if
12: end function
```

LP Formulation Algorithms (provided the LP is certainly bounded)

```
1: function SOLVE-PROBLEM-A(input-to-A)
2:     LP = map input-to-A to a linear program
3:     outcome = SIMPLEX-LP-SOLVER(LP)
4:     if outcome is "infeasible" then
5:         return output in infeasible case
6:     else
7:         output = map LP solution to A solution
8:         return output
9:     end if
10: end function
```

LP Formulation Algorithms (provided the LP is certainly feasible)

```
1: function SOLVE-PROBLEM-A(input-to-A)
2:     LP = map input-to-A to a linear program
3:     outcome = SIMPLEX-LP-SOLVER(LP)
4:     if outcome is "unbounded" then
5:         return output in unbounded case
6:     else
7:         output = map LP solution to A solution
8:         return output
9:     end if
10: end function
```

LP Formulation Algorithms (provided the LP is certainly bounded & feasible)

1: **function** SOLVE-PROBLEM-A(input-to-A)
2: LP = map input-to-A to a linear program
3: outcome = SIMPLEX-LP-SOLVER(LP)
4: output = map LP solution to A solution
5: **end function**

Standard Form versus General Form

general form LP: a "real" LP, more flexible than standard form (previous slides)

Standard Form	**General Form**
maximize objective function	maximize **or minimize** obj. func.
all variables are non-negative	no restriction (may be negative)
every constraint is $\leq$ r.h.s.	constraint may be $=$ or $\geq$ r.h.s.

Standard Form Example

maximize $2x_1 + x_2 - \frac{1}{3}x_3$
subject to

$$\begin{aligned} x_1 + x_2 &\leq 10 \\ -x_3 &\leq -2 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

General Form Example

minimize $x_1 - x_2 + \ldots + c_n x_n$
subject to

$$\begin{aligned} x_1 &\leq 100 \\ x_2 &\geq 2 \\ x_1 + x_2 &= 10 \\ x_2 &\geq 0 \end{aligned}$$

Note

- minimizing objective function
- mix of $\leq, =, \geq$ constraints
- not all variables have $x_i \geq 0$ non-negativity constraint (x_1 is missing)

Formulating General LP as Standard LP

- pre-processing: convert general LP to standard LP
- need to get rid of any non-standard feature
- want insignificant overhead
  - $n = \#$variables, $m = \#$ constraints
  - space complexity of general LP is $\Theta(nm)$
  - want size of resulting standard LP, and time overhead, to both be $O(nm)$
- three features to deal with
  - minimization
  - negative variables
  - $=, \geq$ constraints

Converting "minimize" to Standard Form

- minimize $f(x) \equiv$ maximize $-f(x)$
- so if general LP says

$$\text{minimize } c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

- replace that with

$$\text{maximize } -c_1x_1 - c_2x_2 - \ldots - c_nx_n$$

- size of LP unchanged
- $O(n)$ time to negate coefficients

Converting Negative Variables to Standard Form

The idea:

- if x_j does not have a non-negativity constraint $x_j \geq 0$, call x_j a *negative variable*
- handle each negative variable one at a time
- negative variable x_j becomes two non-negative variables x_j', x_j''
- invariant: $x_j = x_j' - x_j''$
- $x_j' =$ the "positive" part of x_j
- $x_j'' =$ the "negative" part of x_j
- x_j positive $\iff x_j' > 0, x_j'' = 0$
- x_j negative $\iff x_j' = 0, x_j'' > 0$

Converting Negative Variables to Standard Form

The algorithm:

- add non-negativity constraints $x_j' \geq 0, x_j'' \geq 0$
- substitute $(x_j' - x_j'') = x_j$ into LP
  - obj. func: $c_j x_j \Rightarrow c_j(x_j' - x_j'') = c_j x_j' - c_j x_j''$
  - constraints: $a_{i,j} x_j \Rightarrow a_{i,j}(x_j' - x_j'') = a_{i,j} x_j' - a_{i,j} x_j''$
- after solving standard LP, to complete general LP evaluate $x_j = x_j' - x_j''$
- n at most doubles, m unchanged $\Rightarrow$ still $\Theta(nm)$ space
- $O(nm)$ time with care (do all substitutions in one pass)

Converting $=$ Constraints to Inequalities

- this step converts any $=$ constraint to $\leq, \geq$ constraints
- $\geq$ constraints are eliminated in the next step
- $a = b \iff a \leq b$ and $a \geq b$
- replace each $=$ constraint

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n = b_n$$

with two constraints

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n \geq b_n$$

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n \leq b_n$$

- n unchanged, m at most doubles $\Rightarrow$ still $\Theta(nm)$ space
- $O(nm)$ time

Converting $\geq$ Constraints to $\leq$

- $a \geq b \iff -a \leq -b$
- replace each $\geq$ constraint

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n \geq b_n$$

with

$$-a_{i,1}x_1 - a_{i,2}x_2 - \ldots - a_{i,n}x_n \leq -b_n$$

- n, m unchanged so still $\Theta(nm)$ space
- $O(nm)$ time

General LP Problem

general-form linear programming problem
input:

- Boolean for whether f is maximized/minimized
- vector $c \in \mathbb{R}^n$
- vector $b \in \mathbb{R}^m$
- vector $o \in \{\leq, =, \geq\}^m$
- $m \times n$ matrix A of real numbers

output: one of

1. "unbounded";
2. "infeasible"; or
3. "solution" with a vector $x \in \mathbb{R}^n$ maximizing the objective function

Time Complexity

- converting a general-form LP to standard-form takes $\Theta(nm)$ time
- so time to solve general LP is

$$O(nm + \text{time to solve standard LP})$$

- standard-LP solver needs to spend at least $\Omega(nm)$ time just to read in the input
- $\Rightarrow$ time to solve general LP $= O$(time to solve standard LP) (w/ worse constant factors)
- from now on we will formulate general LPs

Formulating General LPs

Need to define

1. the variables, **what they represent**
2. the objective function, whether it is maximize or minimize
3. the constraints, **what they represent**
4. the entire LP in general form
5. the post-processing process
   - say what to do with "infeasible"; **or** prove every LP is feasible
   - say what to do with "unbounded"; **or** prove every LP is bounded
   - solution result; how to interpret solution vector x in terms of business logic

Formulating Business Logic

- business scenario is a "word problem" that defines business-logic rules, resource limits, what needs to be optimized
- you need to figure out how to model all these in parts of an LP
  - variables
  - objective function
  - constraints
- every concept mentioned in the scenario needs to be modeled in the LP somehow
- only works when the objective is a linear function (not squared, exponential, etc.)

Example: Plastic Recycling

Suppose Orange County has the following options for disposing of plastic waste:

Method	**Carbon/ton**	**Dollars/ton**
Local incinerator	2.9	800
Overseas incinerator	3.2	600
Thermal recycling	.5	1200
Landfill	.3	1400

November is expected to produce 700 tons of plastic and is required by law to emit no more than 1400 tons of carbon. The landfill can accommodate up to 100 tons per month and the other methods are effectively unlimited. The County Supervisors want you to minimize the cost of plastic disposal.

Example: Plastic Recycling

1. **Variables:** create a variable to represent how much of each method to use,

$$\begin{aligned} I &\equiv \text{tons incinerated locally} \\ O &\equiv \text{tons incinerated overseas} \\ R &\equiv \text{tons recycled} \\ L &\equiv \text{tons sent to landfill} \end{aligned}$$

2. **Objective function:**
   - "County Supervisors want you to minimize the cost"

$$\text{minimize } 800I + 600O + 1200R + 1400L$$

Example: Plastic Recycling

3. **Constraints:**
   - re-read the word problem and pick out rules
   - "produce 700 tons of plastic":
   $$I + O + R + L = 700$$
   - "no more than 1400 tons of carbon":
   $$2.9I + 3.2O + .5R + .3L \leq 1400$$
   - "landfill can accommodate up to 100 tons":
   $$L \leq 100$$
   - also, use critical thinking, common sense to model implicit rules
   $$I, O, R, L \geq 0$$

Example: Plastic Recycling

4. general-form LP:

minimize $800I + 600O + 1200R + 1400L$
subject to

$$\begin{aligned} I + O + R + L &= 700 \\ 2.9I + 3.2O + .5R + .3L &\leq 1400 \\ L &\leq 100 \\ I, O, R, L &\geq 0 \end{aligned}$$

Example: Plastic Recycling

5. **Interpret results:**
   - infeasible: never happens because there exists at least one feasible solution: $R = 700, I = O = L = 0$
   - unbounded: never happens because the non-negativity constraints lower-bound the objective at 0; it can't approach $-\infty$
   - solution: Tell the Supervisors to incinerate I tons locally, incinerate O tons overseas, recycle R tons, and landfill L tons.

Single-Source Shortest Paths

single-source single-sink shortest paths (1S1SSP) problem
input: weighted undirected graph $G = (V, E)$, source vertex $s \in V$, sink vertex $t \in V$
output: the shortest path $s \rightsquigarrow t$ in G

- Bellman-Ford algorithm solves this in $\Theta(|V| \cdot |E|)$ time
- when every weight $w(e) \geq 0$, Dijkstra's algorithm solves this in $\Theta(|E| + |V| \log |V|)$ time

Formulating Single-Source Shortest Paths

1. **Variables:** for each vertex $v \in V$, create an LP variable

$$d_v \equiv \text{ total weight of shortest path } s \rightsquigarrow v \text{ in } G$$

2. **Objective function:** hold that thought
3. **Constraints:**
   - Recall that in Bellman-Ford and Dijkstra, we compute

$$d_v = \min_{x \in V, \{x,v\} \in E} d_x + w(x, v)$$

   - LP can't express "min" but we can say $d_v \leq$ every candidate:

$$d_v \leq d_x + w(\{x, v\}) \qquad \forall x \in V, \{x, v\} \in E$$

   - $d_s = 0$ because it is the source

Formulating Single-Source Shortest Paths

2. **Objective function:** for sink t, maximize d_t
   - *not* minimize d_t
   - that would allow LP to just set every $d_v = 0$; no notion of edges adding to path weights
   - maximizing d_t forces LP to set each d_v to the $d_x + w(\{x, v\})$ that is the minimum
   - now, when you use an edge, you add its weight
4. **General-form LP:** given $G = (V, E)$, source s, sink t
   create variable $d_v \ \forall v \in V$
   maximize d_t
   subject to

$$\begin{aligned} d_v &\leq d_x + w(x, v) \qquad \forall \{x, v\} \in E \\ d_s &= 0 \end{aligned}$$

Formulating Single-Source Shortest Paths

5. **Interpreting result:**
   - infeasible: never happens because setting every $d_v = 0$ is always feasible
   - unbounded: $d_t = \infty$ means that t is unreachable from s
   - solution:
     - need to identify path (list of vertices) $s \rightsquigarrow t$
     - i.e. identify edges defininining the shortest path and d_t
     - find the constraints where
     $$d_v = d_x + w(x, v)$$
     (not $<$); these vertices x, v are part of the shortest path
     - use BFS or DFS, limited to on-path vertices, to order these vertices

Formulating Max Flow

maximum flow problem
input: a flow network $G = (V, E)$
output: a flow f of maximum value $|f|$

- source $s \in V$, sink $t \in V$
- $c(u, v)$ is capacity, $f(u, v)$ is flow rate of conn. vertices u, v
- **capacity constraint:** $0 \leq f(u, v) \leq c(u, v)$
- **flow conservation:** $\forall u \in V - \{s, t\}$,

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

- value $|f|$ is net flow into sink:

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Formulating Max Flow

1. **Variables:** for each directed edge $(u, v) \in E$, create variable

$$f_{u,v} \equiv \text{flow from } u \text{ to } v$$

2. **Objective function:**
   - want to maximize $|f|$
   - so maximize

$$\sum_{v \in V} f_{s,v} - \sum_{v,s} f_{v,s}$$

Formulating Max Flow

3. **Constraints:**
   - still need to model the **capacity constraint** and **flow conservation** rules of max-flow
   - capacity constraints:

$$f_{u,v} \geq 0 \qquad \forall u, v \in V$$

$$f_{u,v} \leq c(u,v) \qquad \forall (u,v) \in E$$

   (these are separate, not "$0 \leq f_{u,v} \leq c(u,v)$," to match general LP format)
   - flow conservation:

$$\sum_{v \in V} f_{v,u} - \sum_{v \in V} f_{u,v} = 0 \qquad \forall u \in V - \{s,t\}$$

   (using algebra to get constant r.h.s. to match LP format)

Formulating Max Flow

4. **General-form LP:** given flow network $G = (V, E)$, source s, sink t

   create variables $f_{u,v} \quad \forall u, v \in V$

   maximize $\sum_{v \in V} f_{s,v} - \sum_{v,s} f_{v,s}$
   subject to

$$\begin{aligned} f_{u,v} &\geq 0 \qquad \forall u, v \in V \\ f_{u,v} &\leq c(u,v) \qquad \forall (u,v) \in E \\ \sum_{v \in V} f_{v,u} - \sum_{v \in V} f_{u,v} &= 0 \qquad \forall u \in V - \{s,t\} \end{aligned}$$

Formulating Max Flow

5. **Interpreting result:**
   - infeasible: never happens because setting every $f_{u,v} = 0$ is a feasible solution
   - unbounded: never happens because the objective function is

$$\sum_{v \in V} f_{s,v} - \sum_{v,s} f_{v,s},$$

   every $f_{s,v} \leq c(s,v)$, and every $f_{v,s} \geq 0$, so objective is certainly

$$\leq \sum_{u,v \in V} c(u,v)$$

   which is finite
   - solution: define flow function

$$f(u,v) = \begin{cases} f_{u,v} & (u,v) \in E \\ 0 & (u,v) \notin E \end{cases}$$

Placeholder Variables

- *placeholder variable:*
  - variable that represents a concept in the formulation
  - for convenience
  - not strictly necessary
- similar to creating self-documenting variables/constants in coding
- encouraged for clarity
- don't hurt complexity of feasible region $\Leftrightarrow$ not an efficiency concern

Example: mobile phones

- you sell mobile phones for \$99 in USA, and the equivalent price in UK
- your factory in China builds 10,000 phones/month for \$30/unit
- shipping: \$2/unit to UK, \$1.50/unit to USA
- USA charges sales tax = 9% of retail price
- UK charges value added tax (VAT) = 20% of profits (for our purposes)
- goal is to maximize profits, after expenses and taxes
- how many to ship to each country?

First Try: no placeholders

Let $A = \#$ shipped to USA, $K = \#$ shipped to UK
production limit: $A + K \leq 10,000$
non-negative production: $A, K \geq 0$
expenses for USA units: $(30 + 1.5)A = 31.5A$
expenses for UK units: $(30 + 2)K = 32K$
tax for USA units: $.09 \times 99 \times A = 8.91A$
tax for UK units: $.20 \times (99 - 32)A = .20 \times (67)A = 13.4K$
profit:

$$99A + 99K - 31.5A - 32K - 8.91A - 13.4K = 58.59A + 53.6K$$

First Try: no placeholders

maximize $58.59A + 53.6K$
subject to

$$\begin{aligned} A + K &\leq 10,000 \\ A, K &\geq 0 \end{aligned}$$

good: this is a valid LP
bad:

- it's unreadable
- we humans had to do a lot of arithmetic

Second Try: use placeholders

Let

- P_A, P_K, P_T = profits in USA, UK, total, respectively
- S_A, S_K, S_T = sales in USA, UK, total, resp.
- E_A, E_K = production+shipping expenses in USA, UK resp.
- T_A, T_K = taxes in USA, UK resp.

Now:

- objective: maximize P_T
- need constraints to relate the other variables to each other

Second Try: use placeholders

maximize P_T
subject to

$P_T = P_A + P_K$	def'n total profits
$S_T = S_A + S_K$	def'n total sales
$S_T \leq 10,000$	max. production
$S_A, S_K \geq 0$	min. production
$P_A = 99S_A - E_A - T_A$	def'n USA profits
$P_K = 99S_K - E_K - T_K$	def'n UK profits
$E_A = (30 + 1.5)S_A$	USA expenses
$E_K = (30 + 2)S_K$	UK expenses
$T_A = (.09 \times 99)S_A$	USA sales tax
$T_K = (.20(99 - 30 - 2))S_K$	UK VAT

Second Try: use placeholders

good:

- **much** more readable
- easier to formulate (divide-and-conquer)
- maintainable; easy to change one parameter (e.g. shipping cost)

bad:

- longer
- answering business-logic question involves discernment
  - USA sales $= S_A$
  - UK sales $= S_K$
  - ignore all other variables

neutral:

- probably no significant difference in LP solver runtime