60016 OPERATIONS RESEARCH

Introduction to Tutorials

This Lecture

- Real-world LP modelling
  - Resource allocation & blending models
  - Operations planning models
  - Shift scheduling models
  - Time-phased models
- Introduction to GLPK

Real-world LP modelling

- Linear programs have a fairly simple structure:
  - Linear objective
  - Linear constraints
  - Continuous variables
- They all look similar, but the semantics of variables and constraints can be very different.
- Tutorials will train you to model realistic problems using LPs. How to translate textual specifications into LP models?
- It is useful to classify LPs to recognize patterns that arise frequently in real-world applications.

Resource Allocation Models

- The typical goal of a resource allocation model is to split a resource.
- The main issue is how to divide a valuable resource among competing needs.
  - Resources may be capital, land, time,...
- Variables: specify how much of the limited resource is allocated to each use
  - $x_j :=$ area used for growing the jth cereal type
  - $x_j :=$ time used for activity j
  - ...
- Constraints: express constraints on resource availability

$$\sum_j \text{(resource allocation } j) \leq \text{(limit on resource)}$$

Example: Sharing CPU time

x_j:= share of CPU time allocated to application j
c_j:= job completion rate per share unit received by j
s_j:= minimum CPU share that can be assigned to j

Find a share assignment that maximizes the total completion rate.

$$\begin{array}{ll} \text{maximise} & z = c_1 x_1 + \ldots + c_n x_n \\ \text{subject to} & x_1 + \ldots + x_n \leq 1 \ \ (\text{limit on resource}) \\ & x_1 \geq s_1, \ldots, x_n \geq s_n \end{array}$$

Blending Models

- Rather similar to resource models, but focused on combining resources.
- Blending models typically decide what mix of ingredients best fulfills output requirements.
- Variables: specify how much of each ingredient to include in the mix, for example:
  - $x_j :=$ fraction of ingredient j used in the diet
  - $x_j :=$ amount of chemical j used in solution
- Constraints: express the composition of the output

$$\sum_j (\text{\% of property } k \text{ in ingredient } j) \times (\text{amounts of } j \text{ used}) \\ \geq (\text{minimal amount of property } k \text{ in blend})$$

Example: Diet Problem

Determine most economical diet, with basic nutritional requirements for good health.

- n different foods: ith sells at price c_i/unit,
- m basic nutritional ingredients: jth ingredient's daily intake for individual is at least b_j units (healthy diet),
- one unit of food i contains a_{ji} units of the jth ingredient,
- x_i: units of food i in diet (we allow fractional amounts).

(We have a GLPK case study about this.)

Diet Problem (cont)

minimise total cost:

$$z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to:

$$\begin{array}{ccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & \geq & b_1 \\
a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & \geq & b_2 \\
\vdots & & \vdots & & & & \vdots & & \vdots \\
a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & \geq & b_m
\end{array}$$

non-negativity of food quantities

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0$$

Operations Planning Models

- Organizations must decide what to do, when and where.
  - Manufacturing, distribution, government, ...
- Variables: multiple indexes identify products, types of activities, processing facilities, etc.
  - $x_{m,f}$:= amount of material m shipped to factory f
  - $x_{d,f}$:= amount of drink d produced with flavour f
  - $x_{q,v,t}$:= amount of logs of quality q bought from vendor v and peeled with tickness t
- Constraints: balances between inputs and outputs of the activities

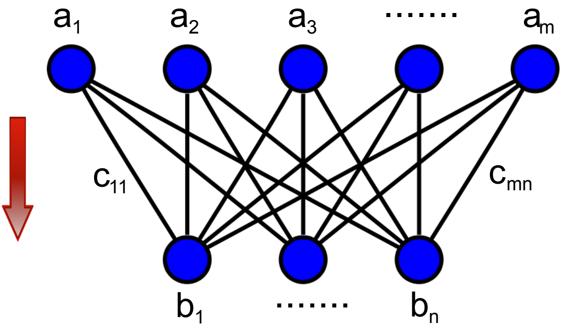
$$(\text{in-flows at stage } i) = \sum_j (\text{out-flows from } i \text{ to stage } j).$$

Example: Transportation Problem

- Quantities $a_1, a_2, \ldots, a_m$ of a product to be shipped from m locations
- Products demanded in amounts $b_1, b_2, \ldots, b_n$ at n destinations
- c_{ij}: unit cost of transporting product from i to j,
- x_{ij}: amounts to be shipped from i to j ($i = 1, \ldots, m$; $j = 1, \ldots, n$). Assume to allow fractional amounts.

Find x_{ij} that satisfy shipping requirements and minimise total cost.

Example: Transportation Problem (cont)



Generalised by the transshipment problem, where products may pass through intermediate nodes (see Tutorial 1).

Shift Scheduling Models

- A class of problems focused on allocating workforce to tasks
- Variables: typically, they indicate a number of employees
  - $x_h :=$ number of employees beginning shift at hour h
  - $x_d :=$ number of part-time employees on shift on day d
- Constraints: allocate workers to cover activities

$$\sum_{s \in \text{shifts}} (\text{output/worker})\ (\text{workers in } s) \geq (\text{output requirement})$$

Shift Scheduling Models (Example)

Assume that shifts are 3 hours:

- $x_h :=$ employees beginning shift at hour h
- $y_h :=$ trainees beginning shift at hour h
- $b_h :=$ minimum number of operators on shift at hour h
- $c :=$ shift pay rate (halved for trainees)

$$\min z = c(x_{11} + x_{12} + x_{13} + x_{14}) + (c/2)(y_{11} + y_{12} + y_{13} + y_{14})$$

subject to:

$$\begin{array}{llll}
x_{11} + y_{11} & & \geq b_{11} & \text{(11:00 shift)} \\
x_{11} + y_{11} + x_{12} + y_{12} & & \geq b_{12} & \text{(12:00 shift)} \\
x_{11} + y_{11} + x_{12} + y_{12} + & x_{13} + y_{13} & \geq b_{13} & \text{(13:00 shift)} \\
\phantom{x_{11} + y_{11} +} x_{12} + y_{12} + & x_{13} + y_{13} + x_{14} + y_{14} & \geq b_{14} & \text{(14:00 shift)}
\end{array}$$

$$\vdots$$

Time-Phased Models

- Time-Phased Models are LPs used to address circumstances that vary over time.
  - Common in cash flow management and scheduling.
- Variables: express returns or state at given time
  - $x_t :=$ projected return on investment by year t
  - $x_{t,p} :=$ projected revenue in week t from sales of product p
  - $x_t :=$ total job completions by time t
- Constraints: time-phase balance constraints

$$\text{(starting level in period } t\text{)} + \text{(impact of period } t \text{ activities)}$$
$$= \text{(starting level in period } t+1\text{)}$$

Time-Phased Models (Example)

- $x_q :=$ cars produced in quarter q (≥ 0)
- $i_q :=$ cars held in inventory at the end of quarter q (≥ 0)
- $d_q :=$ customer demand in quarter q (≥ 0)

(initial inventory) + (product) = (demand) + (ending inventory)

minimise ending inventory: $\min z = i_4$

subject to:

$$0 + x_1 = d_1 + i_1 \qquad \text{(quarter 1)}$$

$$i_1 + x_2 = d_2 + i_2 \qquad \text{(quarter 2)}$$

$$i_2 + x_3 = d_3 + i_3 \qquad \text{(quarter 3)}$$

$$i_3 + x_4 = d_4 + i_4 \qquad \text{(quarter 4)}$$

Other Common Types of LPs

Several other types of LPs exist:

Scenario-based LPs

- Probabilities assigned to different possible situations.

Feasibility problems

- Determine a subset of feasible constraints, for a given infeasible LP.

Computational geometry problems

- Find the smallest convex polyhedron containing given points
- ...

Data envelopment analysis

- Measuring the efficiency of decision making units

$\vdots$

Solving LPs with GLPK

- GLPK is the official linear programming solver by GNU
  - Free and open source (GPL license, written in C)
  - Available on Linux, ported to Windows
- GLPK implements several algorithms we see in class:
  - Linear Programming: Simplex algorithm
  - Integer Linear Programming: Branch-and-Bound, Gomory cuts
- GLPK is fairly similar to solvers used by OR professionals
  - Slower than commercial solvers
  - Language and features very similar to the AMPL+CPLEX commercial suite.
  - AMPL+CPLEX is probably the most popular LP solver around
- Online community and resources: https://www.gnu.org/software/glpk/

Download and Installation

Recommended installation:

- Ubuntu Linux: sudo apt-get install glpk-utils or sudo apt-get install glpk (older versions)

Other ways to get GLPK:

- Linux tarball: http://ftp.gnu.org/gnu/glpk/
- Windows: http://sourceforge.net/projects/winglpk/files/winglpk/
- Windows GUI: http://gusek.sourceforge.net/gusek.html

We always refer to the Ubuntu distribution, but you can use the Windows version if you prefer.

Structure of the Kit

- GPLK available on all DoC machines (let us know if we missed one!)
- Case studies tested with GLPK v4.45
- GLPK v4.45 offers:
  - A command-line linear programming solver (glpsol)
  - The GNU MathProg language (GMPL)
    - Model specification
    - Display and post-processing of results
  - A callable C API (Java's JNI compatible)
  - A lot of nice examples under the installation folder

Example 1: Resource Allocation Problem

Example seen in Lecture 1.

maximise	$y = x_1 + x_2$	:	objective function
subject to	$2x_1 + x_2 \leq 11$	:	constraint on supply of X
	$x_1 + 3x_2 \leq 18$	:	constraint on supply of Y
	$x_1 \leq 4$	:	constraint on demand of A
	$x_1, x_2 \geq 0$	:	non-negativity constraints

Solving Example 1 with GLPK in Two Steps

1. Define a GMPL file specifying the linear program
   - GMPL files are files in plain text with .mod extension
   - Every mod file must be terminated by the keyword end;
   - A mod file may include the keyword solve to explicitly request the solution of the optimization program.
   - Comments are delimited either by /*...*/ or #
   - GMPL syntax available in manuals[1]

```
# example1.mod
var x{i in 1..2}, >=0; /* decision variables */
maximize y : x[1]+x[2];
s.t.
availX : 2*x[1]+x[2] <= 11;
availY : x[1]+3*x[2] <= 18;
demandA : x[1] <= 4;
solve;
end;
```

[1] https://www3.nd.edu/~jeff/mathprog/glpk-4.47/doc/gmpl.pdf

Solving Example 1 in Two Steps

2. Solve the LP: glpsol -m example1.mod -o example1.out
   - Solution saved in example1.out, it is indeed vertex Q=(3,5) from Lecture 1!

```
Problem:      example1
Rows:         4
Columns:      2
Non-zeros:    7
Status:       OPTIMAL
Objective:    y = 8 (MAXimum)
   No. Column name  St   Activity     Lower bound
------ ------------ -- ------------- -------------
     1 x[2]         B              5             0
     2 x[1]         B              3             0
```

GMPL Basics: Parameters

- param: used to inform glpsol about known constant values
- Parameters may be indexed by variables defined over sets
  - A parameter can be indexed over multiple sets
- set: either a range or a finite collection of numbers or strings

```
param m := 3 ;
param n := 2 ;
set M := 1..m ;
set N := 1..n ;
param A {i in M, j in N} ;
param b {j in M} ;
set S := Apple Orange ;
```

GMPL Basics: Data

- Parameters and sets can be specified anywhere, but it is often convenient to group them under the data section
  - The data section is put at the end of mod file or in a separate data file (.dat)

```
# example1.dat
data;
param A :
     1   2   :=
1   2.0  1.0
2   1.0  3.0
3   1.0  0.0  ;
param b :=
1  11.0
2  18.0
3   4.0  ;
end;
```

GMPL Basics: Variables & Constraints

- Variables: decision variables, the unknowns!
  - Variables can be indexed over sets
  - Bounds can also be specified

```
var x {j in N}, >= 0 ;
var u {j in N}, >= 0, <= 3 ;
var w {j in M}, >= 0, <= max {k in M} b[k];
```

- Constraints: set of linear equalities or inequalities
  - Do not need to be in standard form
  - Each constraint must have a label (e.g., "c1" for constraint 1)
  - A set of constraints can be specified by indexing the label

```
s.t.
c1: u[1]+w[1] = 0;
c2{i in M}: sum{j in N} A[i,j]*x[j] <= b[i];
```

GMPL Basics: Objective

- Objective: a linear objective function
  - Either maximize or minimize
  - The objective must have a label

```
maximize z: sum {j in N} x[j];
```

Further Remarks on GMPL

- GLPK allows to index variables using text strings

```
set S := Apple Orange;
var x {i in S}, >=1, <=5; /* units to buy */

    ⋮

s.t. c1: x["Apple"] >= 2*x["Orange"];
```

- Parameters can be assigned a default value
  - Every time glpsol does not have information about a value, it will use the default.
  - Useful for sparse matrices and vectors, you list the non-zero elements and set default 0.0 for the others

```
param cost {i in S} default 1.00;
param matrix default 0.0  :
                 1     2     3     :=
Apple            .     .     .
Orange          1.0    .    2.0
```

Further Remarks on GMPL

- Results can be displayed with printf

```
/* LP */
set S;
param cost {i in S} default 1.00;
var x {i in S}, >=1, <=5; /* units to buy */
minimize z: sum {i in S} cost[i]*x[i];
s.t. c1: x["Apple"] >= 2*x["Orange"];
solve;

/* post-processing */
printf {i in S} "\tx*[%s] = %3.2f\n", i, x[i];

/* data section */
data;
set S := Apple Orange;
end;
```