Optimisation and Operations Research

Lecture 11: Integer Programming

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/notes/OORII/

School of Mathematical Sciences,
University of Adelaide

August 13, 2019

Section 1

Integer Programming Problems

Section 2

Native Integer Variables

Integer variables

A long time ago, in Lecture 1, we looked at our first problem: A manufacturing scheduling problem

$$
\begin{array}{llrcrcrcr}
\max & z = & 13x_1 & + & 12x_2 & + & 17x_3 & & \\
\text{s.t.} & & 2x_1 & + & x_2 & + & 2x_3 & \leq & 225 \\
 & & x_1 & + & x_2 & + & x_3 & \leq & 117 \\
 & & 3x_1 & + & 3x_2 & + & 4x_3 & \leq & 420 \\
\multicolumn{9}{l}{x_1 \geq 0, \; x_2 \geq 0, \; x_3 \geq 0}
\end{array}
$$

$x_1 =$ the number of desks;

$x_2 =$ the number of chairs; and,

$x_3 =$ the number of bed frames, made per time period.

Shouldn't we ensure that x_1, x_2 and x_3 are integers?!

Allocation problem – 1

Example (Knapsack problem)

A hiker can choose from the following items when packing a knapsack:

Item	1 chocolate	2 raisins	3 camera	4 jumper	5 drink
w_i (kg)	0.5	0.4	0.8	1.6	0.6
v_i (value)	2.75	2.5	1	5	3.0
v_i / w_i	5.5	6.25	1.25	3.125	5

However, the hiker cannot carry more than 2.5 kg all together.

Objective: choose the number of each item to pack in order to maximise the total value of the goods packed, without violating the mass constraint.

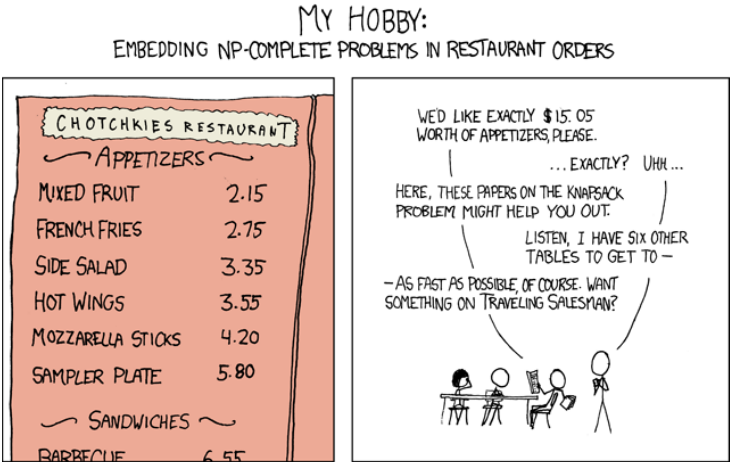
Allocation problem – 2

Example (Knapsack problem – Formulation)

Let x_i bet the number of copies of item i to be packed, such that $x_i \geq 0$ and integer (cannot pack 1/2 a jumper!).

$$\begin{array}{lcccccccccccc}
\max\ v & = & 2.75x_1 & + & 2.5x_2 & + & x_3 & + & 5x_4 & + & 3x_5 & & \\
\text{s.t.} & & 0.5x_1 & + & 0.4x_2 & + & 0.8x_3 & + & 1.6x_4 & + & 0.6x_5 & \leq & 2.5 \\
 & & x_i \geq 0 & & \text{and integer.} & & & & & & & &
\end{array}$$

Allocation problems

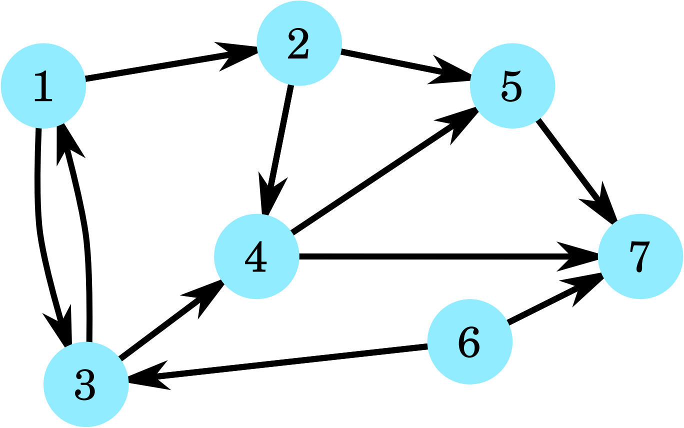


http://xkcd.com/287/

Network Problems

Many optimisation problems are related to a *Network* or *Graph*

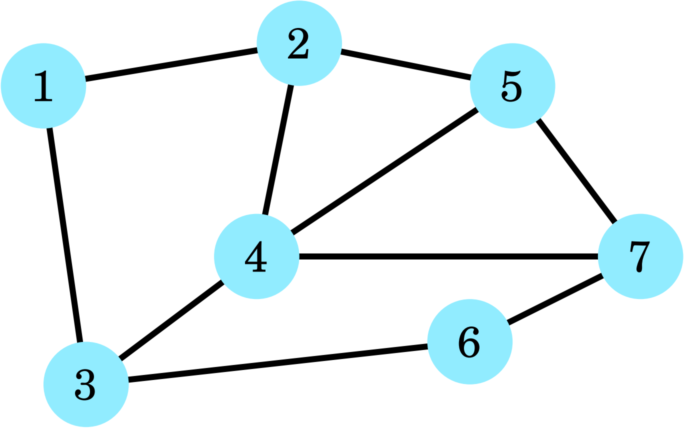
Consider a set of nodes (or vertices) N and a set of directed links (or edges) L between those nodes. These directed links then give us a *directed network* or *directed graph* $G(N, L)$ like that below



Network Problems

Many optimisation problems are related to a *Network*

Consider a set of nodes (or vertices) N and a set of undirected links (or edges) L between those nodes. These directed links then give us an *undirected network* $G(N, L)$ like that below



Graph terminology

Definition (undirected graph)

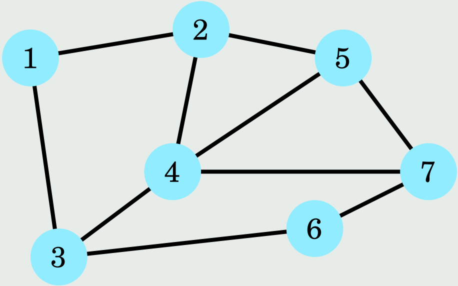
An undirected graph has edges (or links) that are unordered pairs of nodes $\{i,j\} \in L$, $i,j \in N$, *meaning node* i *is adjacent to node* j *and visa versa.*

Definition (directed graph)

A directed graph has edges (or arcs) that are ordered pairs of nodes (i,j), *meaning node* i *is adjacent to node* j.

Graph terminology

Example (Undirected graph)



$G(N, E)$ where N is the set of nodes, and E is the set of edges

$$
\begin{aligned}
N &= \{1, 2, 3, 4, 5, 6, 7\} \\
E &= \{(1,2), (1,3), (2,4), (2,5), (3,4), (3,6), (4,5), (4,7), (5,7), (6,7)\}
\end{aligned}
$$

Graph terminology

- A *walk* is an ordered list of nodes $i_1, i_2, \ldots, i_t$ such that, in an undirected graph, $\{i_k, i_{k+1}\} \in L$, or, in a directed graph, $(i_k, i_{k+1}) \in L$ for $k = 1, 2, \ldots, t-1$.
- A *path* is a walk where the nodes $i_1, i_2, \ldots, i_k$ are all distinct.
  - A graph is *connected* if there is a path connecting every pair of nodes.
- A *cycle* is a walk where the nodes $i_1, i_2, \ldots, i_{k-1}$ are all distinct, but $i_1 = i_k$.
  - A directed graph is *acyclic* if it contains no cycles.
  - we call it a *DAG* = Directed Acyclic Graph

Network Problems

Example (The Travelling Salesperson Problem (TSP))

Given a set of towns, $i = 1, \ldots, n$, and links (i, j) between the towns. The links each have a specified length, given by a distance matrix

$$D = \begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & \cdots & j & \cdots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ n \end{array} & \left[\begin{array}{cccccc} & & & \vdots & & \\ & & & \vdots & & \\ & & & \vdots & & \\ \cdots & \cdots & \cdots & d_{ij} & & \\ & & & & & \\ & & & & & \end{array}\right] \end{array}$$

Objective: construct a directed cycle of minimum total distance going through each town exactly once.

TSP

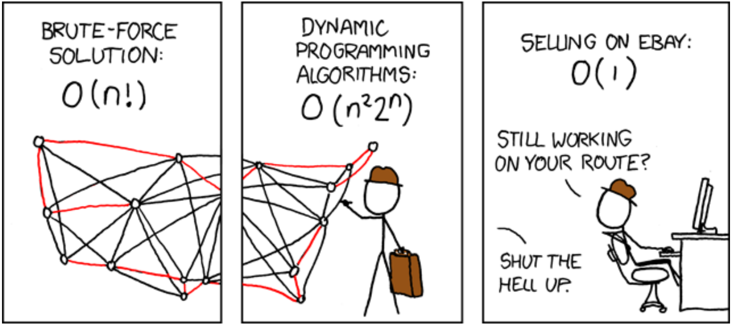
The decision is, basically, which links do we choose to use in the tour.

$$x_{ij} = \begin{cases} 1 & \text{if link } (i,j) \text{ is chosen} \\ 0 & \text{if link } (i,j) \text{ is not chosen} \end{cases}$$

then the ILP (Integer Linear Program) formulation is

$$\begin{aligned}
\min d &= \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} &= 1, \quad \forall i = 1, \ldots, n \quad \text{(only } \textit{one} \text{ link from i)} \\
\sum_{i=1}^{n} x_{ij} &= 1, \quad \forall j = 1, \ldots, n \quad \text{(only } \textit{one} \text{ link to j)} \\
\sum_{i \in S} \left(\sum_{j \in S^c} x_{ij} \right) &\geq 1, \quad \forall S \subset N \quad \text{(connectedness)} \\
x_{ij} = 0 \text{ or } 1 & \quad \text{for all} \quad i, j
\end{aligned}$$

Network Problems



http://xkcd.com/399/

Other Network Problems

Graph and network problems come up a lot!

Example (Shortest path problem)

(Differs from a TSP in that it does not look for a *tour* through all nodes, but rather a path from one node to another.)

Find a minimum length path through a network $G(N, L)$, from a specified source node, say 1, to a specified destination node n, where each link $(i, j) \in L$ has an associated length, d_{ij}.

Example (Maximum flow problem)

Given a network with a single source node (generator of traffic) and a sink (attractor of traffic); network has directed links with links (i, j) having an upper capacity of u_{ij}.

We have no cost for unit flows, just bounds on the capacities of the links and we wish to send the maximum flow from one specified node to another.

Section 3

Supplementary Integer Variables

Supplementary Integer Variables

Sometimes the original variables in the problem aren't variables, but we have to add in extra, artificial variables for some other reason

- disjoint objective functions
  - fixed-cost problems
- disjunctive constraints

Disjoint objective function – fixed-cost problem

Example (Production planning)

Here we have N products, where the production cost for product j ($j = 1, \ldots, N$) consists of a fixed setup cost $K_j \geq 0$ and a variable cost c_j that depends on the number of copies of item j produced. That is, the cost associated with producing x_j copies of item j is

$$C_j(x_j) = \begin{cases} K_j + c_j x_j & x_j > 0, \\ 0 & x_j \leq 0. \end{cases}$$

Objective: minimise total cost, min $z = \sum_{j=1}^{N} C_j(x_j)$, subject to $x_j \geq 0$, and any other constraints on supplies, orders, etc.

Note that variables x_j are not necessarily integer!

Fixed costs problems

The problem is *non-linear*, because of the discontinuity at $x_j = 0$ in $C_j(x_j)$, which can be removed by introducing j new variables y_j, where

$$y_j = \begin{cases} 1 & x_j > 0 \\ 0 & x_j = 0. \end{cases}$$

Then $C_j(x_j) = K_j y_j + c_j x_j$, and the problem can be formulated as

$$\begin{aligned} \min \quad & z = \sum_{j=1}^{N} (K_j y_j + c_j x_j) \\ \text{s.t.} \quad & x_j \geq 0 \\ & y_j = 0 \;\; \text{or} \;\; 1 \\ & x_j \leq M y_j \qquad (+\text{other constraints, } \textit{e.g.}, \text{ order sizes}) \end{aligned}$$

where $M \geq$ upper bound on all x_j.

Fixed costs problems

(a) The original problem didn't have integer variables – these arose as artificial variables to keep the problem linear.

(b) This is a *mixed* Integer, Linear Program *(ILP)*, because some variables are continuous and some are integers.

(c) If $x_j = 0$ then minimising z requires $y_j = 0$.

(d) For all production of item j given by $x_j > 0$ means that we have to set $y_j > 0$ to satisfy the new constraints $x_j \leq My_j$ for each item j.

Disjunctive and conjunctive constraints

Definition

A constraint in the form of A and B is called conjunctive.

Most of our earlier constraints are already conjunctive – we require *all* of them to be true in the feasible region.

Definition

A constraint in the form of A or B is called disjunctive.

These might be used to make it possible to create multiple (separate) feasible regions.

Note, mathematicians sometimes denote AND by $\wedge$ and OR by $\vee$

Disjunctive constraints

Example (Sorta Auto)

Sorta Auto is considering manufacturing 3 types of of cars: compact, medium and large. Resources required and profits yielded by each type of car are

	Compact	Medium	Large
Steel required (tonnes)	0.5	1	3
Labour required (hrs)	30	25	40
Profit ($ 1,000's)	3	5	8

The amount of steel available is at most 2,000 tonnes and at most 60,000 hrs of labour are available. Production of any car type is feasible only if at least 1,000 cars of that type are made.

Objective: maximise profit, by determining how many of each type to produce, while satisfying the constraints.

Disjunctive constraints

Variables

$$\begin{cases} x_1 = \text{number of compact cars produced} \\ x_2 = \text{number of medium size cars produced} \\ x_3 = \text{number of large size cars produced.} \end{cases}$$

Then the objective function is

$$\max \quad 3x_1 + 5x_2 + 8x_3$$

with constraints:

$$\begin{aligned} x_1, x_2, x_3 &\geq 0, \text{ and integer} \\ 0.5x_1 + x_2 + 3x_2 &\leq 2000 \\ 30x_1 + 25x_2 + 40x_3 &\leq 60,000 \end{aligned}$$

$$\begin{array}{llllll} \text{either} & x_1 & = & 0 & \text{or} & x_1 \geq 1000 \\ \text{either} & x_2 & = & 0 & \text{or} & x_2 \geq 1000 \\ \text{either} & x_3 & = & 0 & \text{or} & x_3 \geq 1000. \end{array}$$

Disjunctive constraints

Instead of either $x_i = 0$ or $x_i \geq 1000$ introduce binary variable y_i such that

$$\begin{cases} x_i \leq M_i y_i & \text{(a)} \qquad (M_i \text{ large constant}) \\ 1000 - x_i \leq M_i(1 - y_i) & \text{(b)} \\ y_i \in \{0, 1\} & \text{(c)} \end{cases}$$

Notice that

- if $x_i > 0$, then $y_i = 1$ by both constraints (a) and (c) $\Rightarrow x_i \geq 1000$ because of constraint (b)
- $y_i = 0$ then $x_i = 0$ by constraint (a)

and so we get the desired result!

Disjunctive constraints in general

In general,

"either $f(\mathbf{x}) \le 0$ or $g(\mathbf{x}) \le 0$"

is equivalent to

"if $f(\mathbf{x}) > 0$ then $g(\mathbf{x}) \le 0$".

Introduce $0-1$ variable y and the constraints

$$g(\mathbf{x}) \le M(1-y), \qquad f(\mathbf{x}) \le My$$

where M is a large, positive number.

- if $f(\mathbf{x}) > 0$, then $y > 0$ and so $y = 1$, giving $g(\mathbf{x}) \le 0$.
- if $y = 0$ then $f(\mathbf{x}) \le 0$.

Section 4

Integer Programming: Naïve Solutions

Naïve solutions

1. Exhaustive search
2. Approximate with a LP

Exhaustive search

- When first considering (LP)s we suggested solving them by enumerating all basic solutions, determining which were feasible ones, and then choosing the one which gave the optimal evaluation of the objective function.
  - we saw this was a bad idea (there are too many possibilities)
  - but we have restricted the space now, maybe exhaustive works?
- Approach
  - enumerate every potential solution,
  - check its feasibility, and
  - from amongst the feasible ones, choose the one that gives the optimal value of the objective function.

Exhaustive search

- Consider a simple binary linear program with n binary variables
  - there are 2^n possible solutions
  - that is $O(exp(n))$

  so an exhaustive search is hopeless for even moderate n.
- Consider the TSP
  - N towns to visit
  - $$\frac{(N-1)!}{2} \quad \text{different tours.}$$

    For example, if there were $N = 100$ towns to visit, this is then 4.6663×10^{157} different tours.

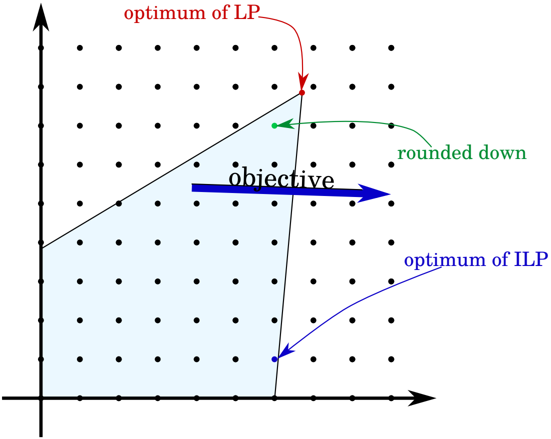
Exhaustive searches are a really bad idea except for toy problems.

Approximate with a LP

1. Idea: drop the integrality constraint and solve the resulting LP
   - we know how to solve LPs efficiently
   - sounds a bit like rounding off, so there might be some errors, but how big can they be?
2. We call this *relaxation*
   - in general relaxation means loosening up some constraint
   - here we relax the integer constraint

Relaxation (of integrality)

We can guess that rounding might not be optimal, but it can be far away from optimal!



It gets worse in higher dimensions.

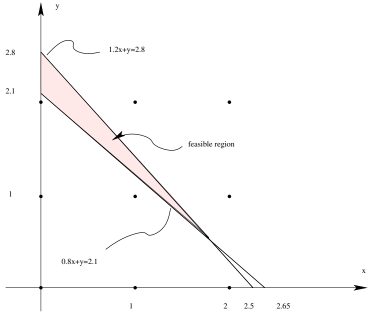
Relaxation (of integrality)

Solving a relaxed (LP) may lead to different solutions

- they *might not even be feasible*

Example

$$
\begin{array}{rrcl}
 & \max\ z & = & 3x + y \\
\text{s.t.} & 0.8x + y & \geq & 2.1 \\
 & 1.2x + y & \leq & 2.8 \\
 & x, y & \geq & 0 \text{ (and } \textit{integral.}\text{)}
\end{array}
$$



So relaxation is a crude tool, but it can be useful if used carefully (we will see how later on).

Less Naïve Solutions

Methods we will look at . . .

- Greedy algorithms

  (heuristics)

- Branch and bound

  (enumeration with pruning)

- General purpose heuristics

  (genetic algorithms)

And we'll look at some toolkits that help.

Takeaways

- Lot's of real problems have *integer* variables
- Integer programming is much more than just linear programming with integer variables.
- Even if variables are continuous, other parts of the problem need extra integer variables
  - disjunctive constraints (either or)
  - discontinuous objectives
- Naïve solutions to ILPs aren't a great idea

Further reading I