Lecture 8: Linear and integer programming modelling and tools

- Normal and standard forms
- Modelling
- Tools

General Linear Programs

- In a general linear program

$$\min \sum_{i=1}^{n} c_i x_i \quad \text{s.t.}$$
$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m$$
$$l_j \leq x_j \leq u_j$$

  inequalities with $\leq$ or $\geq$ can occur in addition to equalities ($=$), maximization can be used instead of minimization, and some of the variables can be unrestricted (do not have bounds).
- A general LP can be transformed to an equivalent (w.r.t. the set of original variables) but simpler form, for instance, to a canonical or standard form (introduced below).
- Two forms are equivalent (w.r.t. a set of variables) if they have the same set of optimal solutions (w.r.t. the set of variables) or are both infeasible or both unbounded.

Standard and Canonical Forms

- An LP is in canonical form when
  - the object function is minimized,
  - all constraints are inequalities of the form $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$, and
  - all variables are non-negative, i.e., bounded by the constraint $x_j \geq 0$.

  that is, the LP is in the form

$$\min \sum_{i=1}^{n} c_i x_i \quad \text{s.t.}$$
$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m$$
$$x_j \geq 0, \quad j = 1, \ldots, n$$

- The standard form is similar but all constraints are of the form $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

Standard and Canonical Forms

An LP can be converted to standard or canonical form using the following transformations:

- Maximization of a function is equivalent to minimization of its opposite: $\max f(x_1, \ldots, x_n) \Leftrightarrow \min -f(x_1, \ldots, x_n)$
- An equality can be transformed to a pair of inequalities

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \Leftrightarrow \begin{cases} \sum_{j=1}^{n} a_{ij} x_j \geq b_i \\ \sum_{j=1}^{n} -a_{ij} x_j \geq -b_i \end{cases}$$

- An inequality can be transfrom to an equality by adding a slack (surplus) variable

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \Leftrightarrow \begin{cases} \sum_{j=1}^{n} a_{ij} x_j + s = b_i \\ s \geq 0 \end{cases}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \Leftrightarrow \begin{cases} \sum_{j=1}^{n} a_{ij} x_j - s = b_i \\ s \geq 0 \end{cases}$$

Transformations—cont'd

- An unrestricted variable x_j can be eliminated using a pair of non-negative variables x_j^+, x_j^- by replacing x_j everywhere with $x_j^+ - x_j^-$ and imposing $x_j^+ \geq 0, x_j^- \geq 0$.
- Non-positivity constraints can be expressed as non-negativity constraints: to express $x_j \leq 0$, replace x_j everywhere with $-y_j$ and impose $y_j \geq 0$.
- These transformations are sometimes needed when modelling if the tool used does not support a feature exploited in the LP model, for example, non-positive or unrestricted variables.

T–79.4201 Search Problems and Algorithms

Example.

- Consider the problem of transforming the LP on the left to standard form. We illustrate the transformation in two steps.

$$\begin{aligned} &\max\ x_2 - x_1 \text{ s.t.} \\ &3x_1 - x_2 \geq 0 \\ &x_1 + x_2 \leq 6 \\ &-2 \leq x_1 \leq 0 \end{aligned}$$

- First: turn maximization to minimization, turn the unrestricted variable x_2 to a pair of non-negative variables and treat bounds as constraints to obtain:

$$\begin{aligned} &\min\ -(x_2^+ - x_2^-) + x_1 \text{ s.t.} \\ &3x_1 - (x_2^+ - x_2^-) \geq 0 \\ &x_1 + (x_2^+ - x_2^-) \leq 6 \\ &x_1 \geq -2 \\ &x_1 \leq 0 \\ &x_2^+ \geq 0, x_2^- \geq 0 \end{aligned}$$

T–79.4201 Search Problems and Algorithms

Example—cont'd

- Second: eliminate non-positivity constraints and transform inequalities to equalities with slack and surplus variables to obtain:

$$\begin{aligned} &\min\ -x_2^+ + x_2^- - y_1 \text{ s.t.} \\ &-3y_1 - x_2^+ + x_2^- - s_1 = 0 \\ &-y_1 + x_2^+ - x_2^- + s_2 = 6 \\ &-y_1 - s_3 = -2 \\ &y_1 \geq 0 \\ &x_2^+ \geq 0, x_2^- \geq 0 \\ &s_1 \geq 0, s_2 \geq 0, s_3 \geq 0 \end{aligned}$$

T–79.4201 Search Problems and Algorithms

Modelling

The diet problem: (a typical problem suitable for linear programming)

- Given
  $a_{i,j}$: amount of the ith nutrient in a unit of the jth food item
  r_i: yearly requirement of the ith nutrient
  c_j: cost per unit of the jth food item
- Build a yearly diet (decide yearly consumption of n food items) such that it satisfies the minimal nutritional requirements for m nutriets and is as inexpensive as possible.
- LP solution: take variables x_j to represent yearly consumption of the jth food item

$$\begin{aligned} &\min\quad c_1x_1 + \cdots + c_nx_n \text{ s.t.} \\ &a_{1,1}x_1 + \cdots + a_{1,n}x_n \geq r_1 \\ &\vdots \\ &a_{m,1}x_1 + \cdots + a_{m,n}x_n \geq r_m \\ &x_1 \geq 0, \ldots, x_n \geq 0 \end{aligned}$$

Knapsack

(a typical problem suitable for (0-1) integer programming)

- Given: a knapsack of a fixed volume v and n objects, each with a volume a_i and a value b_i.
- Find a collection of these objects with maximal total value that fits in the knapsack.
- IP solution: for each item i take a binary variable x_i to model whether item i is included ($x_i = 1$) or not ($x_i = 0$)

$$\begin{aligned}&\max b_1x_1 + \cdots + b_nx_n \text{ s.t.}\\ &a_1x_1 + \cdots + a_nx_n \leq v\\ &0 \leq x_1 \leq 1, \ldots, 0 \leq x_n \leq 1\\ &x_j \text{ is integer for all } j \in \{1, \ldots, n\}\end{aligned}$$

Warehouse Location Problem

(A more complicated 0-1 IP problem)

- There is a set of n customers who need to be assigned to one of the m potential warehouse locations.
- Customers can only be assigned to an open warehouse, with there being a cost of c_j for opening warehouse j.
- Once open, a warehouse can serve as many customers as it chooses (with different costs $d_{i,j}$ for each customer-warehouse pair).
- Choose a set of warehouse locations that minimizes the overall costs of serving all the n customers.
- IP solution: introduce binary variables
  x_j representing the decision to open warehouse j
  $y_{i,j}$ representing the decision to assign customer i to warehouse j

Warehouse Location Problem—cont'd

- Objective function to minimize:

$$\sum_{j=1}^{m} c_j x_j + \sum_{i=1}^{n}\sum_{j=1}^{m} d_{i,j} y_{i,j}$$

- Customers are assigned to exactly one warehouse:

$$\sum_{j=1}^{m} y_{i,j} = 1 \quad \text{for all } i = 1, \ldots, n$$

- Customers can be assigned only to an open warehouse.
  Two approaches:
  - If a warehouse is open, it can serve all n customers:

$$\sum_{i=1}^{n} y_{i,j} \leq n x_j \quad \text{for all } j = 1, \ldots, m$$

  - If a customer i is assigned to warehouse j, it must be open:

$$y_{i,j} \leq x_j \quad \text{for all } j = 1, \ldots, m \text{ and } i = 1, \ldots, n$$

Expressing Constraints in MIP

- Some constraints cannot be represented straightforwardly using linear constraints.
- A frequently occuring situation involves combining constraints "disjunctively".
- An implication is a typical example which can sometimes be encoded by introducing an additional variable and a new large constant.
- **Example.** Consider a binary variable x and the constraint "if $x = 1$ then $\sum_{j=1}^{n} x_j \geq b_i$" where each x_j is non-negative.
  Using a large constant M this can be expressed as follows:

$$\sum_{j=1}^{n} x_j \geq b_i - M(1 - x)$$

  Notice that here if $x = 1$, then $\sum_{j=1}^{n} x_j \geq b_i$ must hold but if $x = 0$, then $\sum_{j=1}^{n} x_j \geq b_i - M$ imposes no constraint on variables $x_1, \ldots, x_n$ if we choose some $M \geq b_i$.

Expressing Constraints—cont'd

- **Example.** Consider a disjunctive constraint "$x \geq 5$ or $y \leq 6$" where x and y are non-negative and $y \leq 1000$.
  This constraint can be encoded by introducing a new binary variable b and constant M as follows

$$x + Mb \geq 5$$
$$y - M(1-b) \leq 6$$

  Here if we choose $M \geq 994$, then
  - if $b = 0$, we have constraints $x \geq 5$ and $y - M \leq 6$ where the latter is satisfied by every value of y ($0 \leq y \leq 1000$) and
  - if $b = 1$, we have constraints $x + M \geq 5$ and $y \leq 6$ where the former is satisfied by every value of $x \geq 0$.
- Unfortunately, these techniques for expressing disjunctions are are not general and, e.g., choosing a value for the constant M is often non-trivial.

Example: Resource Constraints

- In a scheduling application typically following types of variables are used:
  s_j: starting time for job j
  x_{ij}: binary variable representing whether job i occurs before job j
- Consider now a typical constraint:
  "If job 1 occurs before job 2, then job 2 starts at least 10 time units after the end of job 1"
- This is an implication that can be represented by introducing a suitably large constant M (d_1 is the duration of job 1):

$$s_2 \geq s_1 + d_1 + 10 - M(1 - x_{12})$$

  - If $x_{12} = 1$: we get $s_2 \geq s_1 + d_1 + 10$ as required.
  - If $x_{12} = 0$: we get $s_2 \geq s_1 + d_1 + 10 - M$, which implies no restriction on s_2 if M is sufficiently large.

Example: Resource Constraints—cont'd

- Disjunctive constraints on binary variables can be expressed straightforwardly.
- For example, to enforce that the values of variables x_{ij} are assigned consistently according to their intuitive meaning following constraints need to be added.
  - "Either i occurs before j or the reverse but not both"
    This is an exclusive-or constraint which can be encoded directly:

$$x_{ij} + x_{ji} = 1 \quad (i \neq j)$$

  - "If i occurs before j and j before k, then i occurs before k."
    This can be seen as a disjunction $\neg x_{ij} \vee \neg x_{jk} \vee x_{ik}$ of binary variables x_{ij}, x_{jk}, x_{ik}:

$$(1 - x_{ij}) + (1 - x_{jk}) + x_{ik} \geq 1 \text{ (or equivalently } x_{ij} + x_{jk} - x_{ik} \leq 1)$$

    A potential problem: $\mathrm{O}(n^3)$ constraints are needed where n is the number of jobs.

Routing Constraints

(An example of a problem where finding a compact MIP encoding is challenging).

- Consider the Hamiltonian cycle problem:
  INSTANCE: A graph (V, E).
  QUESTION: Is there a simple cycle visiting all nodes of the graph?
- Introduce a binary variable $x_{i,j}$ for each edge $(i, j) \in E$ indicating whether the edge is included in the cycle ($x_{i,j} = 1$) or not ($x_{i,j} = 0$).
- Constraints:
  - The cycle leaves each node i through exactly one edge:

$$\text{for each node } i\text{:} \quad \sum_{(i,j) \in E} x_{i,j} = 1$$

  - The cycle enters each node i through exactly one edge:

$$\text{for each node } i\text{:} \quad \sum_{(j,i) \in E} x_{j,i} = 1$$

Hamiltonian Cycle

- However, the constraints above are not sufficient.
- Consider, for example, a graph with 6 nodes such that variables $x_{1,2}, x_{2,3}, x_{3,1}, x_{4,5}, x_{5,6}, x_{6,4}$ are set to 1 and all others to 0. This solution satisfies the constraints but does not represent a Hamiltonian cycle (two separate cycles).
- Enforcing a single cycle is non-trivial.
- A solution for small graphs is to require that the cycle leaves every proper subset of the nodes, that is, to have a constraint

$$\sum_{(i,j)\in E, i\in s, j\notin s} x_{i,j} \geq 1$$

  for every proper subset s of the nodes V.
- In the example above, this constraint would be violated for $s = \{1,2,3\}$.
- A potential problem for bigger graphs: $\mathrm{O}(2^n)$ constraints needed where n is the number of nodes.

Hamiltonian Cycle–cont'd

- Another approach, where the number of constraints remains polynomial, is to introduce an integer variable p_i for each node $i = 1,\ldots,n$ in the graph to represent the position of the node i in the cycle, that is, $p_i = k$ means that node i is kth node visited in the cycle.
- In order to enforce a single cycle we need to enforce the following conditions.
- Each p_i has a value in $\{1,\ldots,n\}$:

$$1 \leq p_i \leq n$$

- This value is unique, that is, for all pairs of nodes i and j with $i \neq j$, $p_j \neq p_i$ holds.
- For all pairs of nodes i and j with $i \neq j$ such that $(i,j) \notin E$, node j cannot be the next node after i, that is,
  - $p_j \neq p_i + 1$ holds and
  - if $p_i = n$, then $p_j \geq 2$.

Hamiltonian Cycle–cont'd

- For condition 'if $p_i = n$, then $p_j \geq 2$" we can use the technique for implications:

$$p_j \geq 2 - (n - p_i)$$

  Notice that
  - if $n = p_i$, then we get $p_j \geq 2$ and
  - if $n > p_i$, then the constraint is satisfied for all value of p_j ($1 \leq p_j \leq n$).
- To complete the encoding in IP we need to express disequality ($\neq$)

Expressing Disequality

- For expressing an arbitrary disequality $x \neq y$ of two bounded integer variables x and y we reformulate the disequality as "$x > y$ or $y > x$" or equivalently "$x - y \geq 1$ or $x - y \leq -1$".
- Now we can model the disjunction using a binary variable b and a large constant M and the constraints

$$x - y + Mb \geq 1$$
$$x - y - M(1-b) \leq -1$$

  Notice that
  - if $b = 0$, then we get $x - y \geq 1, x - y \leq M - 1$ and
  - if $b = 1$, then we get $x - y + M \geq 1, x - y \leq -1$

  where the constraints involving M are satisfied by all values of x, y given large enough M w.r.t. to the bounds on the values of x, y.

MIP Tools

- There are several efficient commercial MIP solvers.
- Also public domain systems exists but these are not as efficient as the commercial ones.
- See, for example,
  `http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html`
  for MIP systems and other information and frequently asked questions.

MIP Solvers

- A MIP solver can typically take its input via an input file and an API.
- There a number of widely used input formats (like mps) and tool specific formats (`lp_solve`, CPLEX, LINDO, GNU MathProg, LPFML XML, . . .)
- MIP solvers do not require the input program to be in a standard form but typically quite general MIPs are allowed, that is
  - both minimization and maximization are supported and
  - operators "$=$", "$\leq$", and "$\geq$" can all be used.

lp_solve

- In the third home assignment a public domain MIP solver, `lp_solve` is employed.
- See the newest version (5.5) at
  `http://lpsolve.sourceforge.net/5.5/`
- `lp_solve` accepts a number of input formats

Example. `lp_solve` native format

```
min: x1 + x2 + 3x3;
     x1 - x2 <= 1;
     2x2 - 2.5x3  >= 1;
     -7x3 + x2 = 3;
> lp_solve < example
Value of objective function: 3

Actual values of the variables:
x1                             0
x2                             3
x3                             0
```