Linear Programming Problems

COMP4691-8691
School of Computer Science, ANU

1 Problem Transformation

Implement and solve the following problem in PuLP (you can use the `pulp_template.py` file as a starting point):

$$\begin{aligned} \min_{x,y,z} \quad & x + 2y - 5z \\ \text{s.t.} \quad & x + 2y + 3z \geq 2 \\ & y \geq x + 2z \\ & z \leq 6 \end{aligned}$$

Convert the above problem to an equality and non-negative variable form (only equality constraints and non-negative variables). Implement the transformed form in PuLP and verify you get the same solution.

See `transform.py`.

$$\begin{aligned} \min_{x,y,z,s} \quad & x^+ - x^- + 2y^+ - 2y^- - 5z^+ + 5z^- \\ \text{s.t.} \quad & x^+ - x^- + 2y^+ - 2y^- + 3z^+ - 3z^- - s_1 = 2 \\ & y^+ - y^- - s_2 = x^+ - x^- + 2z^+ - 2z^- \\ & z^+ - z^- + s_3 = 6 \\ & x^+, x^-, y^+, y^-, z^+, z^-, s_1, s_2, s_3 \geq 0 \end{aligned}$$

2 Visualising Constraints

Write out the inequalities that represent the feasible region in figure 1. Write two objective functions for this feasible region, one where the problem is bounded and one where it is unbounded.

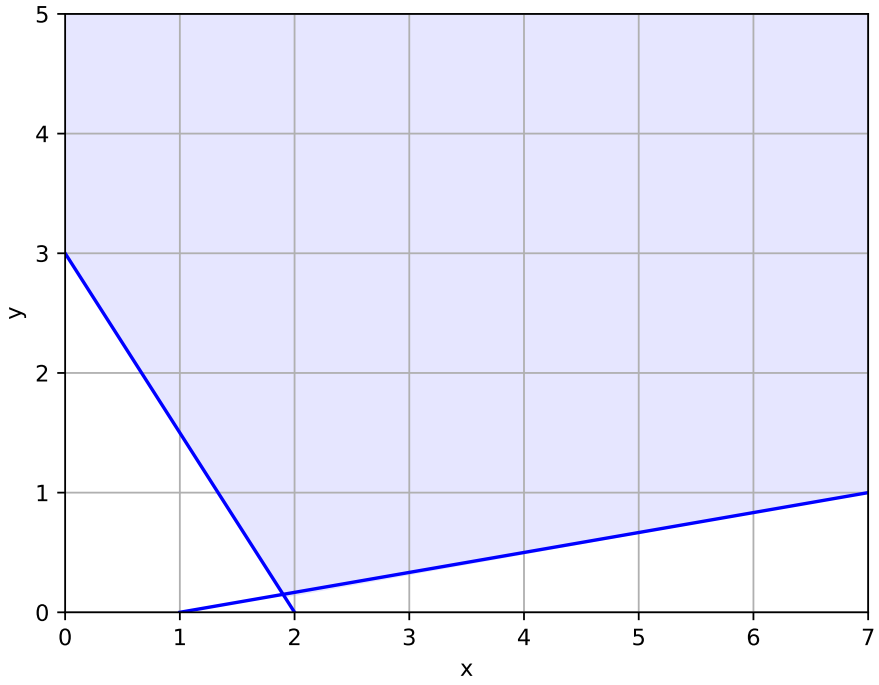

Figure 1: Initial feasible region.

Add the following constraints to the figure:

$$x \geq 1$$
$$4y \geq 5x - 15$$
$$7y + 5x \leq 35$$

Is the modified problem feasible, infeasible, bounded and / or unbounded?

Existing inequalities:

$$y \geq -\frac{3}{2}x + 3$$
$$6y \geq x - 1$$

Assuming a minimisation problem: bounded for $x + y$, unbounded for x. See figure 2 for the new feasible region. The new problem is feasible and bounded for all objectives.

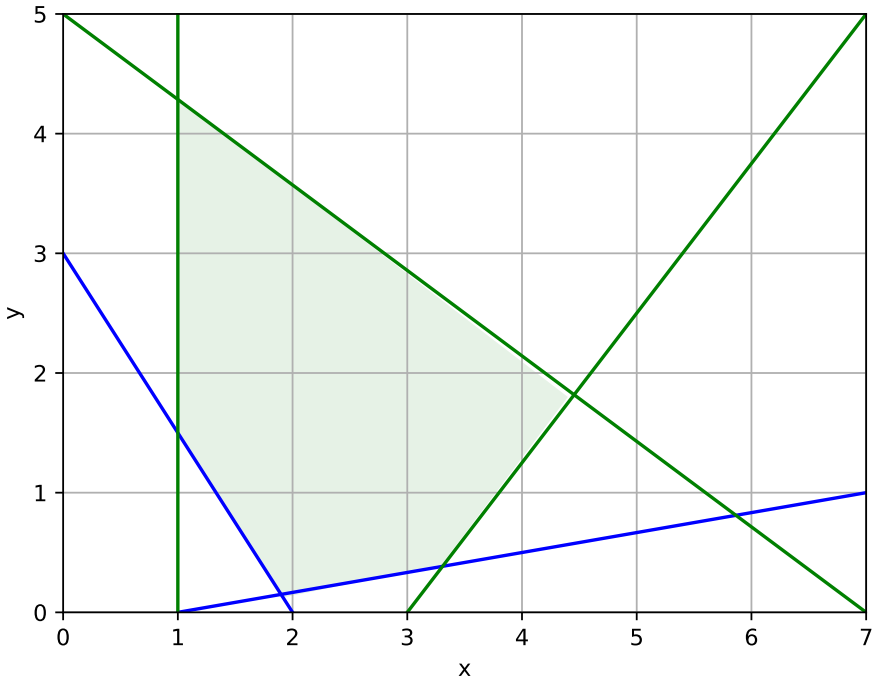

Figure 2: Feasible region with new constraints.

3 Convex Hull

Draw the convex hull of the feasible region in figure 3.

See figure 4.

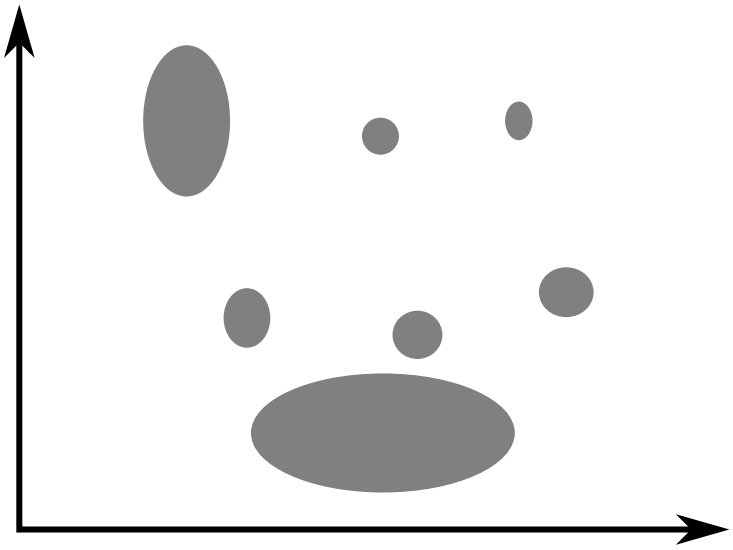

Figure 3: Feasible region.

4 Degeneracy

Explain what degenerate vertices are. Why do they matter for the simplex algorithm?

Degenerate vertices are coincident vertices of the polyhedron that defines the feasible region for an LP. This occurs when different subsets of the linear constraints define the same extreme point.

The simplex algorithm needs to account for degenerate vertices in order to ensure convergence and improve performance. This is done by carefully choosing pivot rules that ensure that the algorithm will not get stuck cycling between degenerate BFSs, so it can continue to make progress improving the objective value.

5 Starting Solution

Explain how a starting feasible solution is obtained for the simplex algorithm.

Alternative simplex:

A modified version of the problem called the feasibility problem is solved in

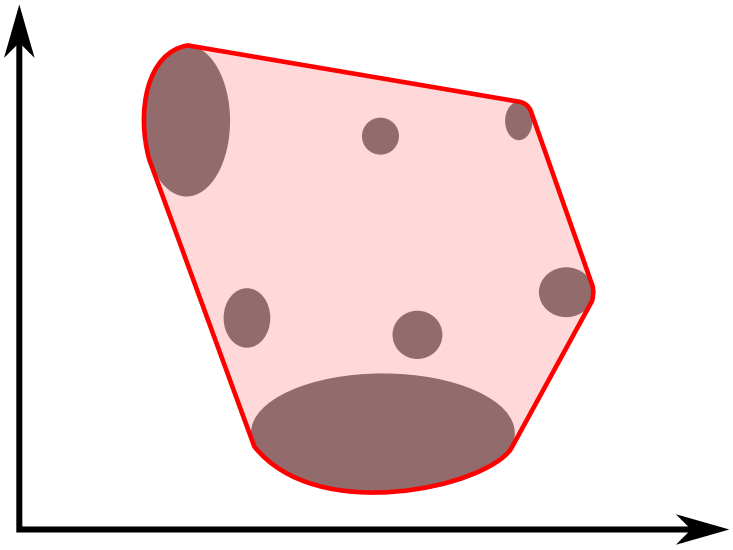

Figure 4: Convex hull of feasible region.

order to find a starting vertex. A slack variable is added to each constraint in order to relax the constraints. The objective is to minimise these slack variables, at which point the original constraints are all satisfied. If this feasibility problem has an optimal solution with one or more non-zero slack variables, then the original problem must be infeasible.

In order to solve the feasibility problem (which is an LP) via simplex we need to find a starting vertex for it. Fortunately there is a trivial starting vertex which can be obtained by solving a linear system for n of the original constraints.

Standard / revised simplex:

A modified version of the problem is solved in order to find a starting BFS. First the problem should be converted into a form with equality constraints and non-negative variables. Non-negative slack variables are then added for each equality constraint. Assumbing b is non-negative (multiply constraint by -1 otherwise):

$$Ax = b \rightarrow Ax + s = b$$

These and the non-negativity constraints then form a new optimisation problem with the objective to minimise the sum of the slack variables. This prob-

lem is started with the trivial BFS with all the slack variables in the basis (and hence all the original variables set to zero). The simplex algorithm is run over this modified problem until an optimal solution is found. If any of the slack variables are non-zero at this point, then the original problem must be infeasible. Otherwise the solution is used as a starting BFS for the original problem.

6 Simplex Pivoting

Explain how a constraint is selected to enter the active set in a pivot step of the "alternative" simplex algorithm. Alternatively, for the standard / revised simplex algorithm explain the analogous operation where a basic variable is chosen to exit the basis.

Alternative simplex:

Once a constraint has been chosen to leave the active set, we want to move along the edge defined by the intersection the $n-1$ constraints that remain in the active set. The constraint that will enter the active set will be the constraint that we first reach as we move along this edge of the feasible region. This is done by factoring in the edge-constraint alignment and the constraint slack.

It is possible that we can move along the edge indefinitely without ever reaching a constraint, in which case the problem is unbounded. It is also possible that we simultaneously reach multiple constraints at the same time in which case a pivot selection rule would be employed.

Standard / revised simplex:

Once an entering variable has been chosen, the simplex algorithm proceeds to finding a suitable exiting variable. This is done by observing which of the existing basic variables will first reach zero as the chosen entering variable is increased away from zero (as it changes from being a non-basic to basic variable).

It is possible that none of the basic variables approach zero as the entering variable is increased, which means that the problem must be unbounded as the entering variable could be increased indefinitely, improving the objective value indefinitely.

Otherwise at least one basic variables will reach zero. If multiple basic variables simultaneously reach zero, then the pivot rule might specify which to pick.

7 Exact Relaxation

What does it mean for a relaxation of a problem to be exact?

An exact relaxation is one which will have the same optimal solution as the original un-relaxed problem.

8 Convex Relaxation

Derive a linear convex relaxation for the following constraints, using at most 3 inequalities:

$$y = ax^2$$
$$0 \leq x \leq \overline{x}$$

where x and y are variables, and $a > 0$. Plot an example of the relaxation. How do you expect the worst-case error in this relaxation to change with $\overline{x}$. Bonus: derive the worst-case error for your relaxation.

Explain how the relaxation could be improved given more inequalities to work with.

There are many ways this could be done. A simple but weak relaxation is:

$$y \leq a\overline{x}^2$$
$$y \geq 0$$

A stronger relaxation is:

$$y \leq a\overline{x}x$$
$$y \geq 0$$
$$y \geq 2a\overline{x}x - a\overline{x}^2$$

An example of this relaxations is shown in figure 5 for $a = 2$, $\overline{x} = 2.5$. The worst-case error will increase for these relaxations as $\overline{x}$ increases.

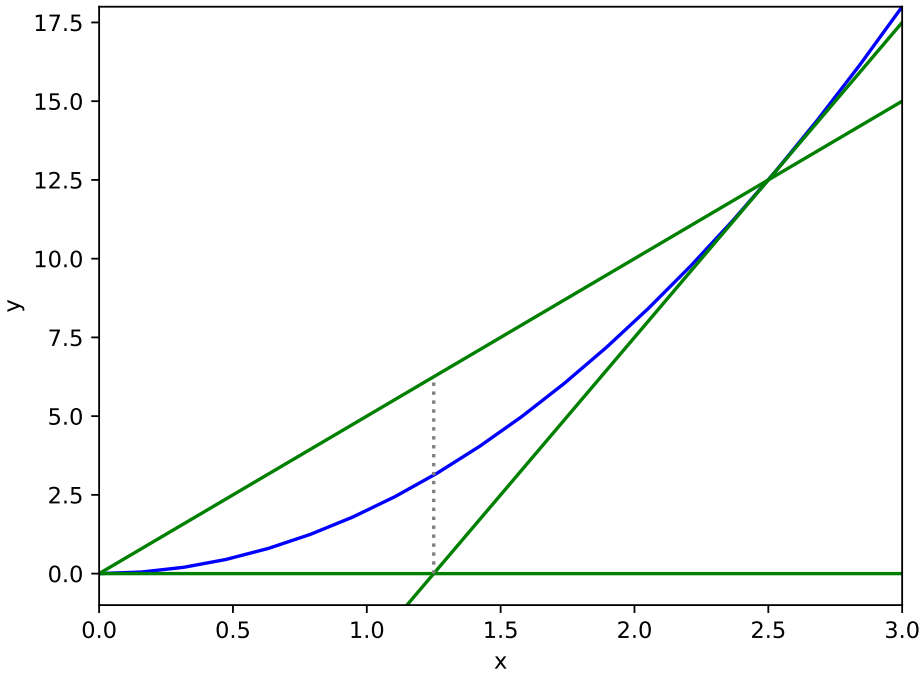

Figure 5: Example of linear convex relaxation for quadratic.

Bonus: the worst-case error above the curve can be found by taking the derivative of the difference between the first inequality and ax^2, setting it to zero and solving for x. The worst-case error below the curve must occur at the intercept of the two lower bounding inequalities. The worst-case error above and below the curve actually ends being at the same x value: $\frac{\overline{x}}{2}$, and the y error at this point is plus or minus $a\frac{\overline{x}^2}{4}$.

If we could add more than the 3 inequalities, the relaxation could be improved by adding more cuts below and tangent to the quadratic curve. Note that first inequality already defines the convex hull above the curve, so could not be improved on given the existing information on the bounds on x and y.

9 Optimality Gap

For a maximisation problem the following values were discovered:

1. A proven lower bound on the objective of 20.
2. Known feasible solutions with objective 10, 15, and 30.
3. A proven upper bound on the objective of 40.

What is the optimality gap for this problem at this point in time?

The optimality gap for a maximisation is the difference between the smallest upper bound and the largest found feasible solution:

$$40 - 30 = 10$$

As a percentage it is $10/30 = 33\%$ (or relative to upper bound $10/40 = 25\%$).

10 Forming the Dual

Transform problem from question 1 into its dual: Implement the dual version in PuLP and verify it gives the same objective value as the primal. Give two other names for dual variables.

$$\begin{aligned} \max_{v_1,v_2,v_3} \quad & 2v_1 - 6v_3 \\ \text{s.t.} \quad & v_1 - v_2 = 1 \\ & 2v_1 + v_2 = 2 \\ & 3v_1 - 2v_2 - v_3 = -5 \\ & v_1, v_2, v_3 \geq 0 \end{aligned}$$

See `dual.py`. Some other names used for duals variables: Lagrange multipliers, shadow prices, marginal prices, reduced costs.

11 Pipe Upgrades

Implement the water pipe upgrade problem presented at the end of the LP 3 Optimality lecture slides, using `water_upgrades.py` as a starting point.

Confirm the solution for the high demand instance with that presented in the lecture slides.

See `water_upgrades.py`.

12 Optimal HVAC Operation

Optimising the operation of heating, ventilation and air conditioning (HVAC) in a building can help reduce energy consumption and electricity costs. This problem is similar to the optimal battery scheduling problem we did in lectures, but where the state variables are the room temperatures and the decision variables are how much to heat each room at each time step over the forward horizon. The objective is to minimise the electricity bill, where power prices can vary over time. Rooms must be kept within the comfortable temperature range when they are occupied (e.g., due to scheduled meeting). A further complication is that heat transfers occur between rooms when there is a temperature difference, and also with the external environment.

We will consider buildings made up of n by n identically sized cube-shaped rooms, e.g., as shown for $n = 3$ in figure 6. The thermal power that flows through a wall with thermal conductance σ is:

$$P_{i,j} = \sigma(T_i - T_j)$$

All the internal walls have identical conductances, which can be different from the external wall conductance that insulates the building form the environment. The floor and ceiling heat transfers are ignored.

Each room has a heater (we will focus only on heating), which can be operated from zero up to a maximum limit.

The change in room temperature depends on the net thermal power transfers (sum of power from all walls and from room heater) and the heat capacity C of the room, and is given by:

$$P_{i,net}\Delta t = C(T_{i,t} - T_{i,t-1})$$

where Δt is the time step length in hours.

The cost of operating the heaters is equal to the energy consumed (the heater power multiplied by the time step length) in each time slot, multiplied by the electricity price for that time step.

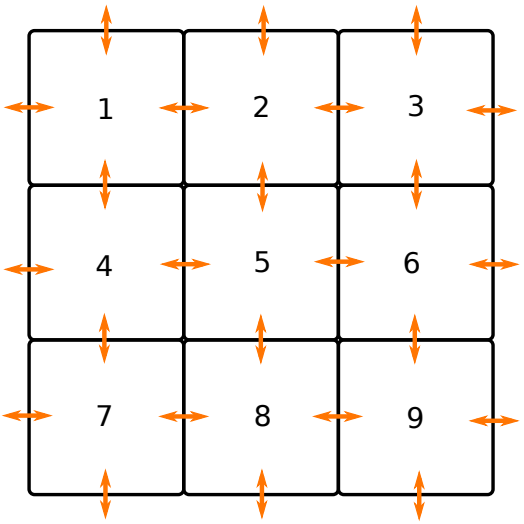

Figure 6: Top-down view of a 3 by 3 building showing wall heat transfers.

Code up your model in `hvac_solver.py`, referring to the files `hvac.py` for the input and output types and `hvac_instances.py` for the two instances.

12.1 Implementation

a) Implement the LP for this HVAC operation problem based on the above description. What are the electricity bills for the two instances?

b) Calculate the dollar savings relative to just keeping all rooms warm (i.e. by forcing temperature limits after first hour).

c) Can you identify any preheating of rooms prior to the periods of occupation? Why might this be occurring?

d) What happens if the heater capacity is limited to 2 kW?

a) See `hvac_solver.py`, and figures 7 and 8. Overall cost is \$3.53, \$65.99 for 1 by 1 instance and 6 by 6 instance respectively.

b) \$4.95, \$93.22 if always occupied after first hour. The savings relative to this are therefore \$1.42 and \$26.23.

c) It occurs for two reasons. 1 it can take multiple time steps to bring the room temperature within range, so heating has to start earlier. 2 it is

occasionally done to take advantage of cheaper electricity prices.

d) For the small instance costs increase to $3.74. The large instance is infeasible.

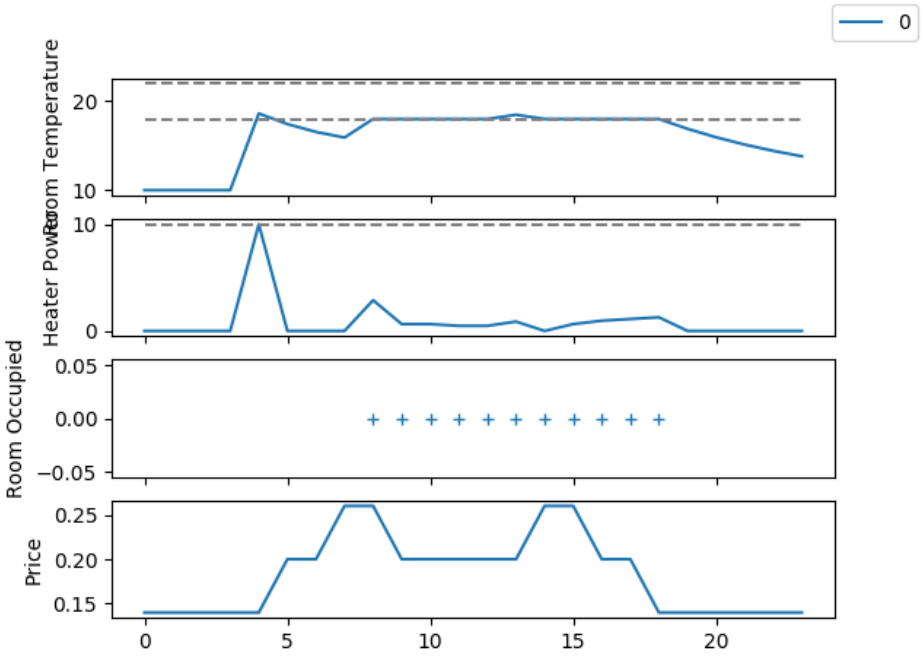

Figure 7: HVAC 1 by 1 instance solution.

12.2 Soft Constraints

The current implementation allows any temperatures between the upper and lower limits when the room is occupied. The occupants have complained that this often leads to temperatures sitting at the lower limit, when in practice occupants prefer the temperature to be in the middle of the range (unless there is a *strong enough* economic / environmental reason to deviate).

This preference can be implemented as a soft constraint. After conducting a survey, it has become apparent that occupants would like a saving of $0.12 per hour for every degree variation in temperature either side of 20°C in each occupied room. The existing hard limits are still in place. Management have

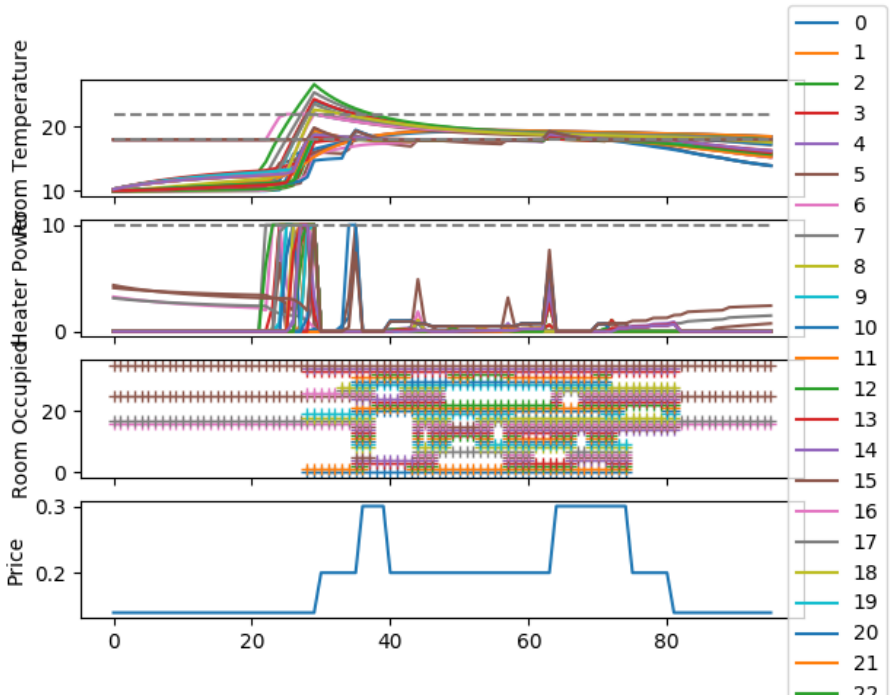

Figure 8: HVAC 6 by 6 instance solution.

agreed to reward occupants by contributing half of the soft constraint cost to the yearly Christmas party.

a) Implement the above soft constraint. What are the electricity bill and the soft constraint cost components for the two instances?

b) Why might management have come up with this way of rewarding customers and will it make sense on their balance sheet?

c) Assuming this day is typical, estimate how much money management will end up contributing to the office Christmas party.

a) $4.42, $83.85 electricity and $0.33, $6.56 for soft constraint.

b) The solution will only deviate from the preferred temperature of 20°C when the savings in electricity costs equal or outweigh the cost associated with the soft constraint. Therefore, when returning half of the soft constraint to the building occupants, management will still save at least that much again in electricity costs, compared to the case of

keeping the temperature at 20°C during times of occupation.

c) For small instance $365 \cdot 0.33/2 = \$60.23$, and for large instance $365 \cdot 6.56/2 = \$1190$.

13 Optimal Power Flow

The optimal power flow (OPF) problem is to determine the cheapest dispatch of generators to meet estimated power consumption, while taking into account the physical flow of power on the network and various operating limits. Electrical flows can be thought of as a type type of *potential* network flow problem, where a difference in potential between two nodes drives the flow.

The equations that govern the flow of power on AC electricity networks are non-linear and even non-convex in nature. However, to a first approximation they can be linearised, which can work reasonably well at the high voltage transmission level:

$$P_{i,j} = -b_{i,j}(\theta_i - \theta_j)$$

Where $P_{i,j}$ represents the power flowing from node i to node j in the network along a transmission line, and θ_i represents the voltage phase angle (in radians) at node i (think of it as a potential). $b_{i,j}$ is a (typically negative) parameter of a line known as the susceptance (similar to conductance).

Each node may have a load and / or generator connected.

Just like regular network flow, the sum of power entering a node (including the power from any generators and loads present at the node) must be equal to zero.

Other constraints include power limits for each line $-\overline{P}_{i,j} \leq P_{i,j} \leq \overline{P}_{i,j}$, and a power limit for each generator $\overline{G}_i$. Other parameters include the load at each node, and the price of operating each generator.

One of the nodes is labelled the reference node, where the voltage phase angle is fixed to zero: $\theta_1 = 0$.

The objective is to minimise the cost of generation.

a) Write an OPF LP based on the above problem description, first mathematically, and then implemented in PuLP using `opf_solver.py`.

Modelling the network as a directed graph (N, E) and assuming there is a load and generator at each node. In addition to the notation introduced above, let G_i be the generation, L_i be the load, and a_i be the price of generation:

$$
\begin{aligned}
\min_{P_{i,j}, G_i, \theta_i} \quad & \sum_{i \in N} a_i G_i \\
\text{s.t.} \quad & P_{i,j} = -b_{i,j}(\theta_i - \theta_j) \quad \forall (i,j) \in E \\
& \sum_{\forall (j,i) \in E} P_{j,i} - \sum_{\forall (i,j) \in E} P_{i,j} + G_i - L_i = 0 \quad \forall i \in N \\
& G_i \in [0, \overline{G}_i] \quad \forall i \in N \\
& P_{i,j} \in [-\overline{P}_{i,j}, \overline{P}_{i,j}] \quad \forall (i,j) \in E \\
& \theta_1 = 0
\end{aligned}
$$

For implementation see `opf_solver.py`. The costs for the two instances are \$4.8k/h and \$2.2M/h.