Mixed-Integer Programming Problems

COMP4691-8691
School of Computer Science, ANU

1 Constraint Encoding

Instead of having to re-implement the MIP encodings of the constraints from the MIP 2 slides, we would like to have a re-usable library of useful constraints. I've started the process in the file `general_constraints.py`, which includes a function `pulp_abs` that takes a pulp model and two variables, and creates the necessary auxiliary variable and adds the required constraints to the model.

Pick one of the constraints from the second half of lecture 6 (MIP modeling), and implement a similar function (try to generalise them even further than I have, e.g., if you pick the or operator, try enabling more n operands).

Create a few tests to check the edge cases of the implementation.

2 Disjunction Constraint

Consider the following constraints:

$$
\begin{aligned}
& ay + bz \geq |x| \\
& x \leq cy \\
& x \geq -dz \\
& x \leq -e \vee x \geq f
\end{aligned}
$$

where $x \in \mathbb{R}$, $y \in [0, \overline{y}]$ and $z \in [0, \overline{z}]$ are variables, and $a, \ldots, f \in \mathbb{R}_{>0}$ are parameters. Reformulate them into mixed-integer linear constraints, avoiding the use of big-M's by deriving appropriate bounds for x.

3 Convex Hull

Explain the significance of the convex hull of a MILP's integer feasible points, and how in general we want a MILP formulation to have a linear relaxation that is as close to this convex hull as possible.

4 Tightening a Formulation

We have a ILP with the following constraints:

$$x + 3y <= 4.5$$
$$8x + 2y <= 23$$

where $x, y \in \mathbb{Z}_{\geq 0}$. Tighten these two constraints without eliminating any integer feasible solutions.

5 Branch and Bound / Cut

Assume we are minimising a MILP using branch and bound. At one point in time in the algorithm we have two open nodes: A and B. The linear relaxation at these nodes have produced objective values of 15 and 30 respectively. The best feasible solution we have found so far has an objective value of 40. Which of the following statements are true (explain why).

a) B has a greater lower bound than A so can be pruned because it cannot possibly contain the optimal solution.

b) We can prune A because the lower bound of B is closer to the current best known feasible solution.

c) There is no point in looking for a better lower bound to B at this point in time.

d) There is no point in looking for a better feasible solution at node B at this point in time.

e) Any children of node B will have linear relaxations with optimal solutions at least as big as that for B.

6 Gomory Cuts

Explain what a Gomory cut is and how they can be used in a branch and cut algorithm.

7 Car Assembly

See the car scheduling example from the end of lecture 7 (MIP branch-and-bound). This has been partially implemented in `files/assembly.py`. Only the constraints on the use of the robot arms have been implemented so far (for the first part we will ignore the ordering, elevation and mutex constraints). The implementation is currently performing poorly: it can't prove it has found an optimal solution within 60s.

For each question, report the solve time (capped at 60s), the optimal solution (or best found + lower bound if timed out), the number of enumerated nodes, and the value of the continuous linear relaxation at the root node (scroll to beginning of the solver output until you find the "Continuous objective value is" entry). You might also want to store the entire solver output printed to the screen for each part for later reference when comparing the solver performance.

a) Run the starting file, and report the performance. A makespan of 37 is indeed the optimal solution to this restricted version of the problem. What does the solver output after 60s suggest about the strength of our formulation?

b) Add in new variables that indicate whether or not each job is running at each time point. Link these with the original start time indicator variables, and modify the arm limit constraint to take advantage of these new variables. If I claim this formulation isn't any stronger than the original, what could explain the improved performance?

c) Add variables that take on the starting time of each job. Link these to the original start time *indicator* variables, and update the makespan constraints to take advantage of them. How has this stronger formulation helped performance?

d) Add in job ordering constraints. Does the objective value of the new solution suggest our previous solutions would have also satisfied these

ordering constraints?

e) Add in elevation and mutex constraints. How many times does the solution switch between elevated and lowered during the makespan? Write out a constraint(s) that would allow you to penalise the number of times the car switches state between elevated and lowered (no need to implement).

f) What if we had 4 robot arms available, how does the solution / performance change? What happens when you now remove the elevation constraints?

g) What happens if you reduce the number of time steps used by the model to 30 for the 4 arms version?