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:

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

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

Introducing a binary variable $u \in \{0, 1\}$, the constraints become:

$$\begin{aligned}
&ay + bz \geq x \\
&ay + bz \geq -x \\
&x \leq cy \\
&x \geq -dz \\
&x \geq f + (\underline{x} - f)u \\
&x \leq \overline{x} + (-e - \overline{x})u
\end{aligned}$$

where the bounds are given by:

$$\begin{aligned}
\underline{x} &= \max(-a\overline{y} - b\overline{z}, -d\overline{z}) \\
\overline{x} &= \min(a\overline{y} + b\overline{z}, c\overline{y})
\end{aligned}$$

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.

The convex hull of a MILP is the smallest convex set that includes all integer feasible points. This is a convex polyhedron, where all vertices will have integer feasible points (otherwise it wouldn't be the smallest possible convex set). Just like for LPs, the linearity of the objective will drive optimal solutions to the boundary of the feasible region, and as such there will always be a vertex that is optimal if an optimal solution exists for the problem.

If an MILP's formulations linear relaxation is equal to its convex hull, then all we will need to do is solve an LP in order to solve the overall problem.

If it is not equal to the convex hull but close to, then the solution to the LP relaxation is more likely to return results near to the integer feasible optimal. When this is the case we will have a better chance of obtaining a good or optimal integer solution using simple heuristics such as a feasibility pump.

The solution of the LP will also provide a better lower bound on the integer feasible optimal if it is close to the convex hull. Good lower bounds are critically important for proving whether or not we have found an optimal solution, and for use in techniques such as branch and bound for pruning part of the search space.

4 Tightening a Formulation

We have a ILP with the following constraints:

$$x + 3y \leq 4.5$$
$$8x + 2y \leq 23$$

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

If we just consider each constraint separately, we can tighten them to:

$$x + 3y \leq 4$$
$$4x + y \leq 11$$

However, we can go further by consider both constraints in conjunction:

$$y \leq 1$$
$$x + y \leq 2$$

An example of these getting progressively tighter, for the respective linear relaxations:

- The original allow $y = 4.5/3$ as a solution
- The independent tightening would allow $y = 4/3$ as a solution
- The combined tightening would allow at most $y = 1$ as a solution

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.

a) False: We can't compare lower bounds between nodes to make this decision. We would need to have the lower bound of B greater than the current feasible solution of 40 in order to be able to prune it.

b) False: Similarly we would need the lower bound of A to be greater than 40 in order to prune it.

c) False: There could be a benefit in the doing this, e.g., we could find a lower bound greater than 40 (e.g., by generating cuts), that would then enable us to prune B.

d) False: It is probably not the best strategy at this point in time but it still could turn out beneficial. E.g., if we find a better feasible solution than 40 this might then come in handy later on. Also if we find a value of 30 we would then be able to prune B.

e) True: Branching breaks the parent's feasible region up into parts, and likewise the feasible region of its linear relaxation. The union of the feasible regions of the children linear relaxations will also typically be smaller than that of B. Because the linear relaxation feasible regions are smaller, their optimal solutions must be at least as big as their parent node.

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?

a) 60s best 37, lb 8, root 5. If the optimal is indeed 37, this suggests that our formulation is weak given that it is having troubling closing the lower bound.

b) 15s optimal 37, root 5. Due to the heavy reliance of MIP solvers on a range of heuristics, it can sometimes be difficult to determine exactly why a change improved performance. In this case it could be that the new running indicator variables are providing the solver with better values to branch on, or make it easier for the solver to generate strong cuts. Compared to the original formulation, the new formulation finds the integer feasible solution of 37 slightly faster, but more significantly once it does so it almost instantly closes the optimality gap by generating a number of cuts to improve the lower bound.

c) 2.6s optimal 37, root 16.66. The root node continuous relaxation provides a much better lower bound than before. Also it seems like fewer cuts need to be generated to improve the lower bound than previously. Another significant improvement is that the solver can now very quickly find a feasible solution (under 1s compared to around 8s for the previous question).

d) 9s optimal 37, root 18.2. Even though the makespan is the same, it doesn't mean that the previous solutions satisfied the ordering constraints.

e) 5.52s optimal 37, root 18.2. 7 times for me (may vary). See the CNC miller job sequencing problem for a way of modelling this kind of penalty.

f) 50s optimal 25, root 14.38. After removing he elevation constraints the optimal slightly reduces to 24.

g) For me it didn't prove the optimal in 60s (best 25, lb 23). This goes to show that doing some something that initially seems sensible, like (almost) halving the number of variables and constraints, is not guaranteed to bring a benefit. In some cases doing something like this might make it harder to find an initial feasible solution and therefore delay the branch and bound process (as we need both good lower bounds and feasible solutions to prune). This could be the case here, or it might just be related to the heuristics of the solver.