Integer Linear Programs

Thursday, April 29, 2021 8:05 AM

Following ch. 23 in Robert Vanderbei's book
(For more details, see Nemhauser + Wolsey '88)

An integer linear program (ILP) is a LP in which the variables are constrained to be integers. An ILP is (in general) not convex, but we can exploit its structure to solve it to find a global solution

... the catch: our technique often works quickly but you can find worst-case examples where it's extremely slow. Afterall, ILP are NP-Hard

There are variants which can be treated in similar ways,

eg.
- Mixed-Integer LP's (MILP)
  meaning some variables are integers, others can be any $\mathbb{R}$ number
- Binary LP's
  Each variable is in $\{0,1\}$ or $\{-1,1\}$, a special case of ILP's
- (mixed) Integer Quadratic Programs
- ...

ILPs and variants have excellent commercial solvers (most free for academics) CPLEX, Gurobi, Xpress, Mosek
so if you can recast your non-convex program as an ILP (or variant) that's great.

Ex: $\min_{x \in \mathbb{R}^n} \|x\|_0$ s.t. $Ax = b$ ($\|x\|_0$ = # nonzero entries (not a true norm))

Recast as MILP

$\min_{x \in \mathbb{R}^n, z \in \{0,1\}^n} 1^T z = \sum_{i=1}^n z_i$ s.t. $Ax = b$, $|x_i| \le M \cdot z_i$ $(i = 1, \dots, n)$

Sometimes adding this constraint is called the "big-M technique"

i.e., $z_i = 0 \Rightarrow x_i = 0$, an indicator variable

Branch-and-Bound one of several standard techniques to solve ILP and variants

Example problem:

$$\max_{\vec{x} = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^2} \quad 17 \cdot x + 12 \cdot y \quad \text{(O)}$$

$\mathbb{Z} = \{\text{integers}\}$

$$x, y \in \mathbb{Z}$$

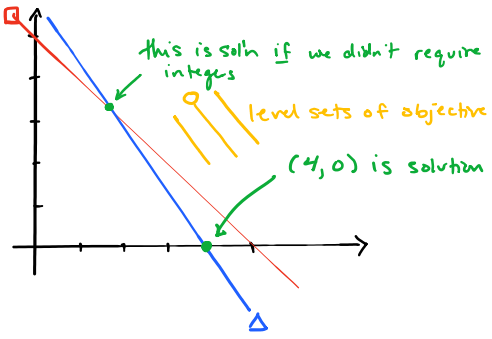
$$x, y \ge 0$$

s.t.

$$10x + 7y \le 40 \quad (\triangle)$$

$$x + y \le 5 \quad (\square)$$

no fundamental difference if min or max since everything is linear



Let p^* be the optimal value of this ILP

First step: solve a relaxation of the ILP where we ignore the integer constraints
(thus just a LP, so convex, "easy" to solve)

Call this solution (x°, y°) w/ optimal value p°

- If the solution vectors happen to be integers, you're in luck! You've solved the ILP.
  - usually very unlikely
  ... but in some network flow problems, it actually happens

- If not, at least we have an upper bound: $p^* \le p^\circ$

(why? we relaxed constraints, i.e., larger feasible set

$\max_{x \in C} \le \max_{x \in C'}$ if $C \subseteq C'$)

Ex: $(x^\circ, y^\circ) = (5/3, 10/3)$, $p^\circ = 205/3 \approx 68.33$ } we know $p^* \le 68.33$

($5/3, 10/3$ not integers.)

Next step: branch ($x^\circ = 1.6$)

Solve two relaxations,

either $x \le 1.6$ or $x > 1.6$

($\Rightarrow x \le 1$ since $x \in \mathbb{Z}$; $\Rightarrow x \ge 2$)

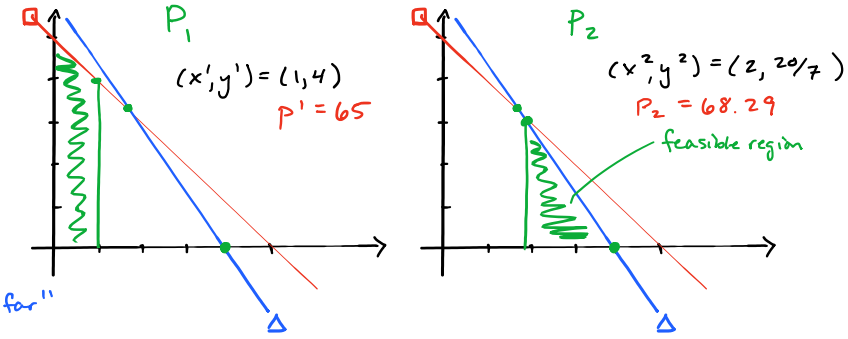
Now more likely to get integral solutions

P_0 : original LP (w/o $\mathbb{Z}$ constraints)

branch

P_1 : add $x \le 1$ P_2 : add $x \ge 2$

(still w/o $\mathbb{Z}$ constraints)



- Soln to P_1 is integral! So this is a feasible pt. to the ILP, hence

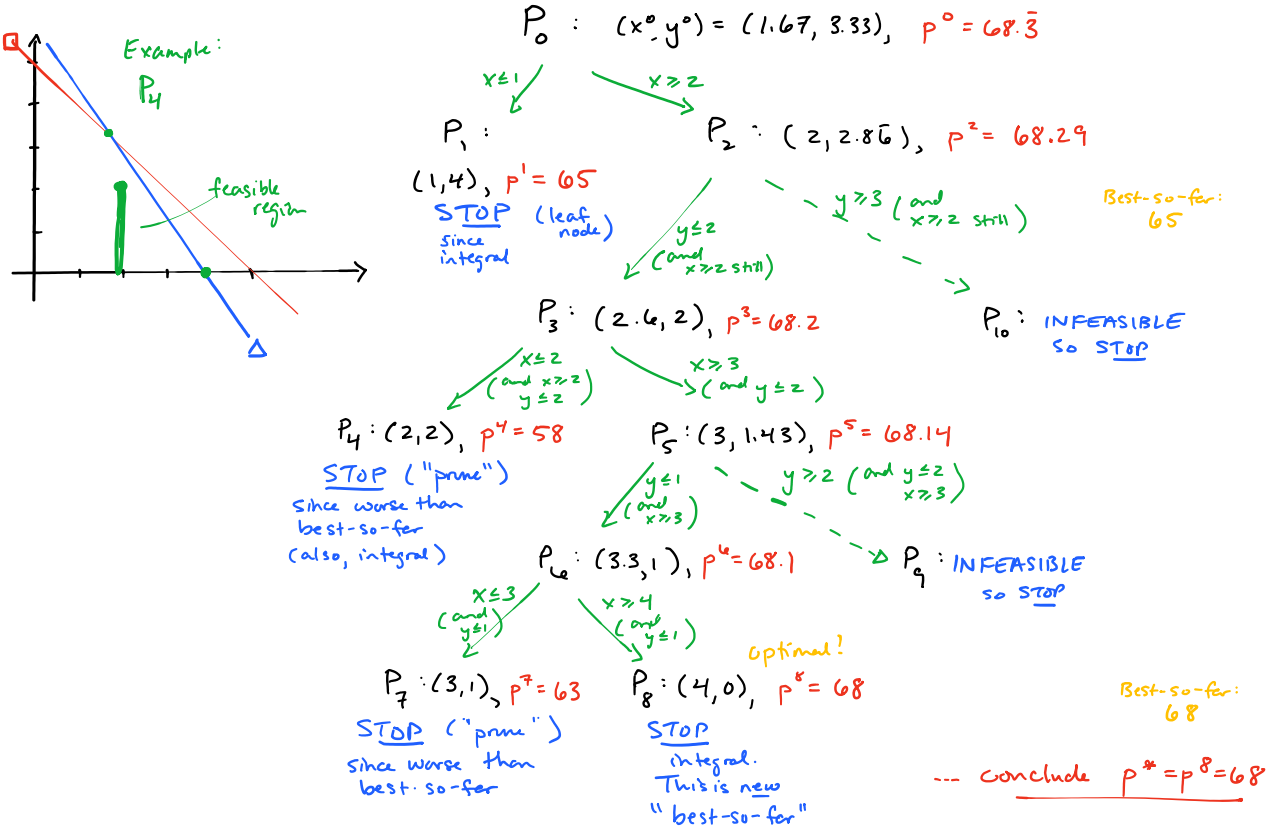
$68.\bar{3} = p^\circ \ge p^* \ge p' = 65$ (bounded!)

Record $p' = 65$ as "best-so-far"

- Soln to P_2 isn't integral, it's $(20, 20/7)$ w/ $p^2 = 68.29$

Observation: had p^2 been less than p', we could have stopped and declared (x', y') solve the original ILP

- Since P_2 not integral, keep branching: see tree below



There's a finite number of possible nodes, so this will terminate, but obviously could be very slow.

But often you get to prune off entire branches

"prune" a branch if ① integral solution
② worse than "best-so-far"
③ infeasible

So... order in which we solve the tree matters

Good software uses good heuristics to determine order

Vanderbei suggests, as a rule-of-thumb, that depth-first is preferable over breadth-first since

1) deep in tree gives integer soln, and
   (a) it's nice to have at least a feasible sol'n to give the user,
   (b) gives "best-so-far" values that are helpful for pruning
2) easy coding via recursion
3) parent-child problems are similar, so have good starting points

Due to the structure of the algorithm, we might find an optimal sol'n quickly, but still need time to verify it

Ex. ILP w/ 10,000 variables, might find optimal soln in 1 day of computing, then need 2 more days of computing to verify it

(so software might write "best so far" to disk in case you decide to terminate the program before it can prove optimality)

Many business problems are MILP
ie "operations research"
-- hence good, expensive software.

Ex: Scheduling airline routes and staff
Scheduling classrooms at a University