**LP**

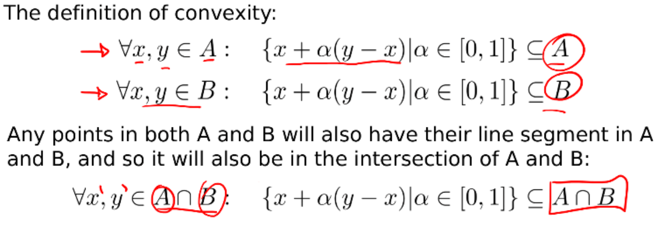
**Decision Vars:** These are variables we control, and they directly influence the solution, **Parameters***:* Constants in the problem that affect outcomes but aren't controlled directly. **Constraints:** Define the limits on decision variables, relationships between variables, and relationships between variables and parameters. **Objective:** Defines what makes a solution "good" by establishing a goal—either maximizing or minimizing a criterion. Model Based Adv: Flexibility: Formulate once and reuse, Automate, easy to update, Disadv: slower than tailored algo, limited control over soln procedures. Algo - problem-specific and often rely on heuristics, which are simplified rules to guide solutions – heuristics and meta heuristics, limited adaptability (new cstr often require rewriting or adjustm).

Leonid Kantorovich39, Defines an LP as a constrained optimization problem with a linear objective function and linear constraints with real valued continuous variables. Can be solved in polynomial time, handling millions of variables and constraints, cannot represent combinatorial choices directly, but often provide proximations.

PuLP is a Python library for LP and MILP, linking with solvers like Clp and Cbc. **LpVariable**: Creates decision variables. **LpProblem**: Sets up the optimization problem i.e. model m. Constraints are added to the model, and then the objective. Status (infeasible as -1, often due to conflicting requirements or overly restrictive constraints), var values, objective value can be printed out. PULP_CBC_CMD, msg = 1 to make the solver more verbose.

**Active Constraints** (or binding constraints) are those that are satisfied as equalities in the optimal solution, meaning they limit the feasible solution directly. **Contract Modification –** Removing a pesky binding constraint by adding a penalty for breaking the contract. If yields higher profit, financially favourable to break the contract.

Negative of obj turns into to min/max problem. Standard form of LP requires **equality constraints** and **non-negative variables,** former is done so using slack variables. Sl vars are non-negative basic variables to emulate an equality and is free to vary. They are used as loose variables which are tightened to loosen a decision var in the obj. **Feasible Region**: The total set of all feasible solutions, also known as the solution space.



The definition of convexity:

$$\forall x, y \in A: \quad \{x + \alpha(y - x) | \alpha \in [0, 1]\} \subseteq A$$

$$\forall x, y \in B: \quad \{x + \alpha(y - x) | \alpha \in [0, 1]\} \subseteq B$$

Any points in both A and B will also have their line segment in A and B, and so it will also be in the intersection of A and B:

$$\forall x, y \in A \cap B: \quad \{x + \alpha(y - x) | \alpha \in [0, 1]\} \subseteq A \cap B$$

The **convex hull** of a set of points is the smallest convex region containing all those points, often represents the feasible boundary in LP, helping to define the area within which all potential solutions exist. An **extreme point** of a convex set S does not belong to an open line segment for **any** two points in S.

**Unbounded Solution**: When the feasible region extends indefinitely, leading to potentially infinite values for the objective. **Single Optimal Solution**: Occurs when the objective function touches the feasible region at a single point, like a vertex. **Multiple Optimal Solutions**: Found along an edge or face where the objective function aligns (slope of obj = slope of the edge). **Infeasible Solution**: If the feasible region is empty, no solution exists. For obj $c^T x$, for minimization, we move in the opposite direction of c.

Since the objective function is linear, it defines a direction in which its value increases/decreases. To solve the problem graphically, all we must do in this direction and record the last intersection, which is the optimum, which will be achieved in at least one vertex. We can move from vertex to vertex, which is called pivoting, picking one that brings us closer to the goal.

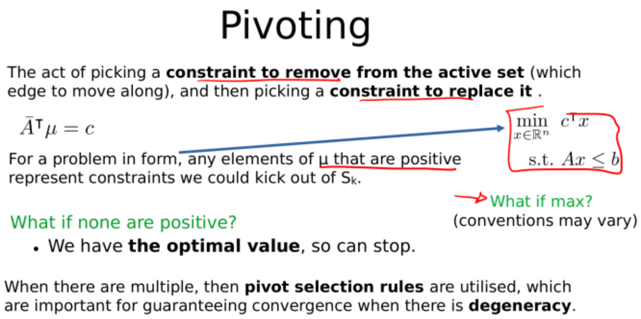
The **Simplex algorithm** is a key method for efficiently solving LP problems by systematically navigating the vertices of the feasible region to locate the optimal solution (not polynomial time). The crucial idea about simplex algorithm: **Choose the Entering Variable**: Look at the objective function row and pick the variable with the largest positive coefficient. This choice maximizes the increase in the objective. **Determine How Far to Move**: For each constraint involving this variable, calculate how far you can go before hitting a limit. **Select the Exiting Variable**: The slack variable to tighten, based on the largest non-positive ratio between the entering variable and the constants. **Keep doing this all the coefficients in the objective fn are negative**. Opposite for minimisation problem.

**Standard Simplex**: Uses a tableau form, which is accessible but best suited for smaller examples.

**Revised Simplex**: Operates with matrix manipulation, reducing memory and computational load, making it ideal for larger LP problems. **Dual Simplex**: Allows for adjustments in feasibility while maintaining optimality, which is useful for dynamically changing constraints.

A vertex in R$^n$ is defined by the intersection of **n** linearly independent hyperplanes (associated with n equality or active inequality constraints). **Step 1**: Select n linearly independent constraints from the m constraints available. **Step 2**: Solve the resulting n×n system to locate an intersection point, a candidate for a vertex. **Step 3**: Confirm that this point satisfies all constraints to ensure its within the feasible region. The number of potential vertices grows exponentially with problem size, making exhaustive checking impractical. Simplex optimizes by focusing only on feasible vertices that lead toward the optimal solution and if **there are no neighbouring vertices that improve the objective, then our current vertex is optimal.**

**Degeneracy** occurs when multiple vertices share the same coordinates due to overlapping constraints, potentially causing Simplex to revisit the same solution.  Degeneracy can lead to **cycling**, where the algorithm repeatedly encounters the same points, slowing or even preventing convergence. **Pivot Rules** like **Bland's Rule** ensure convergence by choosing pivots that prevent revisiting points, helping the algorithm move past degenerate vertices.



Pivoting

The act of picking a **constraint to remove from the active set** (which edge to move along), and then picking a **constraint to replace it** .

$\bar{A}^\intercal \mu = c$

For a problem in form, any elements of μ that are positive represent constraints we could kick out of Sₖ.

$$\min_{x \in \mathbb{R}^n} c^\intercal x \quad \text{s.t. } Ax \le b$$

What if max?

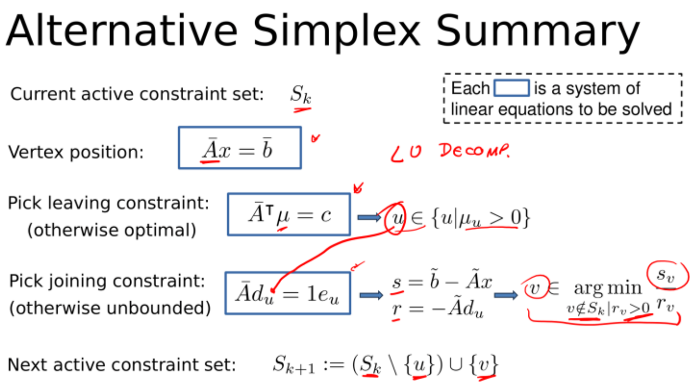
(conventions may vary)

What if none are positive?

- We have **the optimal value**, so can stop.

When there are multiple, then **pivot selection rules** are utilised, which are important for guaranteeing convergence when there is **degeneracy**.

Efficiently, Improving the obj value.



Alternative Simplex Summary

Each ▭ is a system of linear equations to be solved

Current active constraint set: $S_k$

Vertex position: $\boxed{\bar{A}x = \bar{b}}$

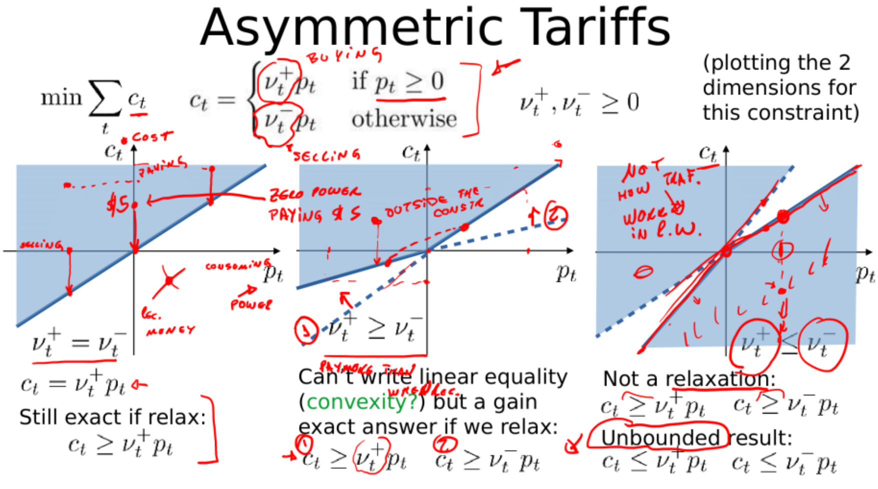
Pick leaving constraint: (otherwise optimal) $\boxed{\bar{A}^\intercal \mu = c} \Rightarrow u \in \{u | \mu_u > 0\}$

Pick joining constraint: (otherwise unbounded) $\boxed{\bar{A}d_u = 1e_u} \Rightarrow \begin{matrix} s = \tilde{b} - \tilde{A}x \\ r = -\tilde{A}d_u \end{matrix} \Rightarrow v \in \underset{v \notin S_k | r_v > 0}{\arg\min} \frac{s_v}{r_v}$

Next active constraint set: $S_{k+1} := (S_k \setminus \{u\}) \cup \{v\}$

A **basis** is a set of linearly independent columns from the LP constraints matrix, representing the current active constraints in a BFS. **Variables in Basis**: Basic variables correspond to the basis, while non-basic variables are set to zero in the current solution. Start at an initial bfs, entering, leaving and then **pivot**: Update the solution based on the new basis.

A relaxation is an approximation where we **increase the feasible set.** Can be done cstr to cstr, like x>=10 to x>=9. **Relaxation** involves loosening some constraints of a problem to make it more tractable, usually by converting non-linear or integer constraints into linear, continuous forms. **Convex Relaxation**: A specific form of relaxation where non-convex constraints are transformed into a convex set, allowing for LP or convex optimization techniques. Conversely, we can restrict the feasible set by creating a subset with tightened points or tightening constraints, i.e. cutting off chunks of feasible region like x+y<=10 to x+y<=8. **Exact Relaxation:** The modified problem can have same objective value and exact solution (same optimal var values). i.e. min fx or arg min fx is equal to modified set X'. Inactive constraints can be removed or tightened to become active i.e. removing the slack.



Asymmetric Tariffs

$$\min \sum_t c_t \qquad c_t = \begin{cases} \nu_t^+ p_t & \text{if } p_t \ge 0 \\ \nu_t^- p_t & \text{otherwise} \end{cases} \qquad \nu_t^+, \nu_t^- \ge 0$$

(plotting the 2 dimensions for this constraint)

$\nu_t^+ = \nu_t^-$

$c_t = \nu_t^+ p_t$

Still exact if relax:

$c_t \ge \nu_t^+ p_t$

$\nu_t^+ \ge \nu_t^-$

Can't write linear equality (convexity?) but a gain exact answer if we relax:

$c_t \ge \nu_t^+ p_t \qquad c_t \ge \nu_t^- p_t$

$\nu_t^+ \le \nu_t^-$

Not a relaxation:

$c_t \ge \nu_t^+ p_t \qquad c_t \ge \nu_t^- p_t$

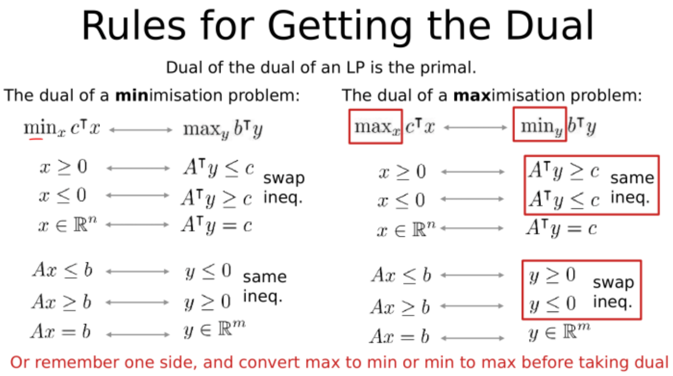
Unbounded result:

$c_t \le \nu_t^+ p_t \qquad c_t \le \nu_t^- p_t$

P is the total power, when positive charging, as in buying power from the grid. Second case is when it costs us more to buy than to sell. We must minimise C i.e. y axis in the feasible region. It's also nonconvex, due to not being affine. So we relax the constraints to be intersection of regions above the lines. Third region is not a relaxation because I can't recover the optimal soln. Second is unbounded since we are trying minimize c.

Battery efficiency, denoted by **η** (eta), measures the fraction of energy effectively stored or discharged. For charging, the internal power **b̃** is less than or equal to the external power multiplied by the efficiency: **b̃ ≤ η * b** For discharging, the reverse applies (due to efficiency losses): **b̃ ≤ b / η**. **Roundtrip Efficiency**: the efficiency over a full cycle of charging and discharging is calculated as the square of η: **Roundtrip efficiency = η^2** For example, if η (efficiency) is 90%, then the roundtrip efficiency is 0.9 * 0.9 = 0.81, or 81%. **Non-Const Efficiency:** By adding constraints at multiple points along the efficiency curve, the model can approximate the true behaviour of the battery's efficiency more accurately, without resorting to complex nonlinear equations. **The optimality gap** is the difference between the upper and lower bounds of the solution range in a minimization problem. It indicates higher confidence i.e. how close the solution is to the true optimum. The optimality gap for a maximisation is the difference between the smallest upper bound and the largest found feasible solution, inverse for minimisation.

**Dual:** Scaling factor for each primal constraint. The dual problem aims to find the largest lower bound for the primal objective, constrained by inequalities that preserve the primal-dual relationship.



Rules for Getting the Dual

Dual of the dual of an LP is the primal.

The dual of a **min**imisation problem:

$\min_x c^\intercal x \longleftrightarrow \max_y b^\intercal y$

$x \geq 0 \longleftrightarrow A^\intercal y \leq c$ swap

$x \leq 0 \longleftrightarrow A^\intercal y \geq c$ ineq.

$x \in \mathbb{R}^n \longleftrightarrow A^\intercal y = c$

$Ax \leq b \longleftrightarrow y \leq 0$ same

$Ax \geq b \longleftrightarrow y \geq 0$ ineq.

$Ax = b \longleftrightarrow y \in \mathbb{R}^m$

The dual of a **max**imisation problem:

$\boxed{\max_x} c^\intercal x \longleftrightarrow \boxed{\min_y} b^\intercal y$

$x \geq 0 \longleftrightarrow A^\intercal y \geq c$ same

$x \leq 0 \longleftrightarrow A^\intercal y \leq c$ ineq.

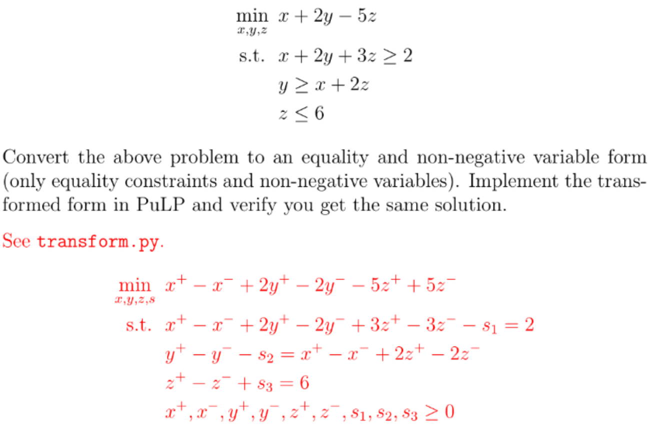
$x \in \mathbb{R}^n \longleftrightarrow A^\intercal y = c$

$Ax \leq b \longleftrightarrow y \geq 0$ swap

$Ax \geq b \longleftrightarrow y \leq 0$ ineq.

$Ax = b \longleftrightarrow y \in \mathbb{R}^m$

Or remember one side, and convert max to min or min to max before taking dual



$$\min_{x,y,z} \; x + 2y - 5z$$
$$\text{s.t.} \; x + 2y + 3z \geq 2$$
$$y \geq x + 2z$$
$$z \leq 6$$

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`.

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

When a constraint is non-binding, its dual variable is zero. Constraints with higher dual variable values have a stronger impact on the objective, indicating where resources are most valuable.

How a starting feasible solution is obtained for Simplex: *Alternative simplex*: **Create a Feasibility Problem**: This involves modifying the original problem by adding slack variables to relax the constraints. **Objective**: Minimize these slack variables to bring the system into feasibility. If the optimal solution to this feasibility problem has any non-zero slack variables, it indicates that the original problem is infeasible. **Finding sv:** The feasibility problem allows for a straightforward starting point by solving for a trivial starting vertex. *Standard/Revised Simplex*: **Convert to Standard Form**: Transform the problem into equality constraints with non-negative variables. **Introduce Slack Variables**: Add non-negative slack variables to handle equality csrts. The goal is to minimize the sum of the slack variables. Start with a BFS by setting all original variables to zero and using the slack variables in the basis. If all slack variables are zero at the optimal solution, the original problem is feasible, and this solution serves as the starting BFS.

**Pivoting in alternative simplex method**, when a constraint is selected to leave the active set, the algorithm aims to move along the edge defined by the intersection of the remaining n−1 active constraints. As we progress along this edge, the constraint that will enter the active set is the first one encountered. Determining this requires factoring in the alignment between the edge direction and the new constraint, as well as considering the slack available for the movement along this edge. There are two special cases to consider. First, if it's possible to move indefinitely along the edge without encountering a new constraint, the problem is deemed unbounded. Second, if multiple cstr are reached simultaneously, a pivot selection rule is employed to decide which constraint should enter the active set.

The Simplex algorithm typically requires a **basic feasible solution** as a starting point. A basic feasible solution is an extreme point of the feasible region. Since **Alpha-Feas** only guarantees that xα is feasible (it satisfies Ax≤b) but doesn't guarantee it's a basic feasible solution, xα may not be suitable as a starting point for the Simplex algorithm, and it's not guaranteed it would be an extreme point either.

To solve this optimization problem, we begin with the conditions that x−7y≥0, x≥40, and y≥10, aiming to minimize x+5y. Dual objective function is max 40y1 + 10y2 + 0y3 with constraints y1 + y3 <= 1 and y2 - 7y3 <= 5. By simplex, substituting into the objective function, it simplifies to 120 - 70s1 - 10s2 - 30y1, so y1=0, maximize the other two.

**MIP**

MIP combines the flexibility of linear programming (LP) with integer constraints, allowing us to solve more complex problems that require some decisions to be whole numbers (like assigning people or resources in fixed units). The Knapsack Problem is a MILP prob involving selecting items with given values and weights to maximize total value within a weight limit, with a binary variable attached representing whether an item is chosen (1) or left out (0).

**Linear Relaxation**: Eases integer constraints to create a simpler LP model, since ILP is NP-Hard while LP is P. **Convex Hull**: The convex hull of a MILP is the smallest convex set that includes all integer feasible points, forming a convex polyhedron where all vertices correspond to integer feasible points. The strength of a formulation depends on how closely it approximates the **convex hull** of feasible integer solutions. If the linear relaxation perfectly matches the convex hull, any LP solution would also solve the ILP optimally. In other words, a strong convex hull formulation provides a tight relaxation, meaning the feasible region of the relaxed LP is close to the integer solution space. This alignment reduces the search area for integer solutions, improving efficiency. Formulations will typically have to weaken the more general they become. Unfort, the **convex hull changes** not just as the class of problem changes, but also **as parameter values change.**

Types of Cstrs**: Choice constraints** often require binary decisions, where a variable is set to either 1 (choice made) or 0 (choice not made) eg, In the knapsack problem, each item has a binary choice constraint. **Logical constraints** are used to express dependencies or conditions between variables eg: z implies x v y. **Disjunctive constraints** allow one of multiple constraints to hold, which is valuable for modelling choices where only one of several conditions can apply. **Piecewise Linear Constraints:** Useful for approximating nonlinear relationships by breaking them into linear segments. **On-off constraints** control whether certain activities or resources are active (on) or inactive (off) based on decision variables i.e. for u=1, x=0 as in low(x)(1-u) ≤ x ≤high(x)(1-u). The **Big-M** method provides a large constant (M) to enforce conditions indirectly by linking variables. If M is too large, it can worsen the relaxation quality, making solutions less efficient. In cases where we can't easily derive bounds, M is used i.e. x E (-M, M).

Nonlinear relationships (like squared or mod terms) are harder to model directly, so using auxiliary variables to approximate these functions, transforming nonlinear constraints into piecewise linear or other ILP-friendly forms.

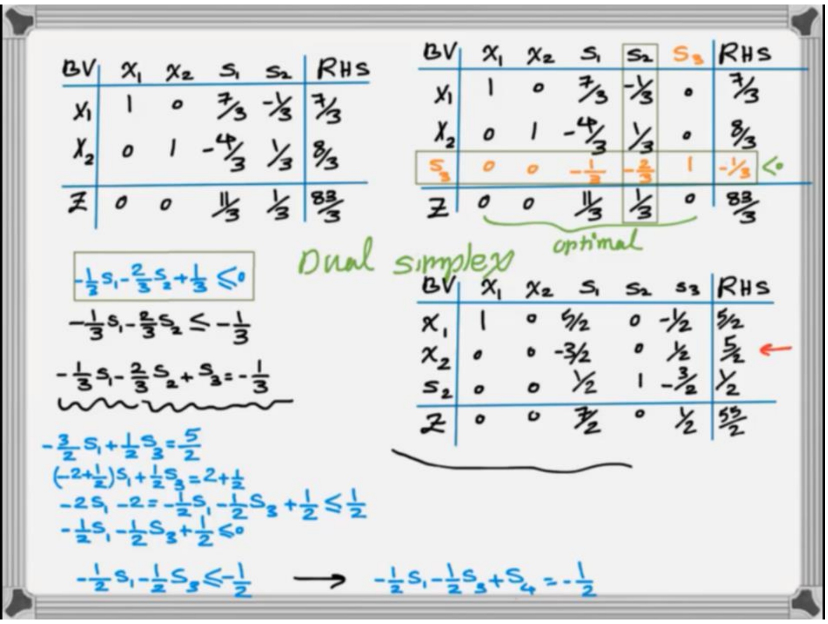
**Branch & Bound** – B&B is a method for systematically exploring the solution space. It divides (or "branches") the problem into subproblems and calculates bounds to eliminate (or "prune") sections that cannot contain the optimal solution. For Milp where dec var are integers, simplex won't help to find a soln. Let's say for a linear relaxation, we calculate the obj value, and the exact values, but they are no longer feasible if they are fractional. Starting the branch and bound with the soln (i.e. the upper bound) with first subprob with the linear soln, picking the var with the largest decimal portion e.g. x2 = 2.67, branch out with <= 2 or >=3 constraints, since x2 cannot be between 2 & 3 since that's not a feasible answer. Taking the linear relaxation again for every node, repeat until all branches are pruned. For max prob, the best current feasible answer is the lower bound (because we assume there might be another solution that beats this solution), and the upper bound of a node is the maximum objective value over its entire region. If the feasible region of a node is empty, it is pruned. A branch is fathomed if the lb of a node is lower than the current LB. If the UB of a node is lower than the current LB, that branch is pruned. For min, for each partition, a lower bound is calculated by solving the linear relaxation of the partition, if an integer feasible solution is found within a partition, it becomes an upper bound for the overall problem. Because each node presents a restricted feasible region therefore its lower bound is not a lower bound of the original problem. Since the open nodes represent a partition of the original feasible region, the lowest lower bound is a valid lower bound for the original problem

The optimality gap is the difference between the best feasible integer solution found so far and the lowest bound among the unexplored nodes. If the gap is within an acceptable range, the algorithm may stop, considering the solution sufficiently close to optimal. Strategy: Feasibility heuristics quickly find feasible integer solutions to provide a good upper bound, reducing the search tree size by pruning nodes that cannot yield better solutions. **S**olve the LP relaxation to get a fractional solution. **R**ound this solution to the nearest integer. **P**roject back to the feasible region to reduce constraint violations. Repeat until a feasible integer solution is found or a set number of iterations is reached. Feasibility pump play is used to find feasible solutions that may not necessarily be optimal.

**Scheduling Constraints -** Start Time Variables: s_j represents the start time of job j (Suitable for continuous

time representation and precedence constraints but can complicate resource modeling). Indicator Variables: x_jt = 1 if job j starts at time t, and 0 otherwise, with time discretized into intervals (Suitable for modelling resource constraints by representing each period but requires time discretization and more variables). s_j >= s_i + d_i, where d_i, can be used to set deadlines. X4 <= x5 – d4 + 12u, x4>=x5+d5-12(1-u), where u dictates whether job 4 comes before or after. For indicator, like y3 <= sum of t for y2, upto t-d2-d3, i.e. to ensure no otr job.

**Gomory Cut – 1Step:** Linear Relaxation, **2Step:** Add new constraints called the cutting cstrs, to cut the feasible region and get best f integer soln. For that, solve simplex, pick one of the basic variables with fractional values and write eqn from the optimal table e.g. x1 + 7/3s1 = 1/3. Rewriting the equation so the fractional values are written as the summation of the integer value and a decimal portion e.g. x1 + (2+1/3)s1 = 1/3 -> x1 + 2s1= -1/3s1+1/3. For RHS, since slack are non-negative, that means -1/3s1 +1/3<=1/3, rounding rhs to integer, -1/3s1 + 1/3<=0 – cutting eqn.



| BV | $x_1$ | $x_2$ | $s_1$ | $s_2$ | RHS |
|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | $7/3$ | $-1/3$ | $7/3$ |
| $x_2$ | 0 | 1 | $-4/3$ | $1/3$ | $8/3$ |
| $Z$ | 0 | 0 | $11/3$ | $1/3$ | $83/3$ |

$$\boxed{-\frac{1}{3}s_1-\frac{2}{3}s_2+\frac{1}{3}\leq 0}$$

$$-\frac{1}{3}s_1-\frac{2}{3}s_2\leq -\frac{1}{3}$$

$$-\frac{1}{3}s_1-\frac{2}{3}s_2+s_3=-\frac{1}{3}$$

| BV | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | RHS |
|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | $7/3$ | $-1/3$ | 0 | $7/3$ |
| $x_2$ | 0 | 1 | $-4/3$ | $1/3$ | 0 | $8/3$ |
| $s_3$ | 0 | 0 | $-\frac{1}{3}$ | $-\frac{2}{3}$ | 1 | $-1/3$ |
| $Z$ | 0 | 0 | $11/3$ | $1/3$ | 0 | $83/3$ |

optimal

Dual simplex

| BV | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | RHS |
|---|---|---|---|---|---|---|
| $x_1$ | 1 | 0 | $5/2$ | 0 | $-1/2$ | $5/2$ |
| $x_2$ | 0 | 0 | $-3/2$ | 0 | $1/2$ | $5/2$ |
| $s_2$ | 0 | 0 | $1/2$ | 1 | $-3/2$ | $1/2$ |
| $Z$ | 0 | 0 | $7/2$ | 0 | $1/2$ | $55/2$ |

$$-\frac{3}{2}s_1+\frac{1}{2}s_3=\frac{5}{2}$$

$$(-2+\frac{1}{2})s_1+\frac{1}{2}s_3=2+\frac{1}{2}$$

$$-2s_1-2=-\frac{1}{2}s_1-\frac{1}{2}s_3+\frac{1}{2}\leq\frac{1}{2}$$

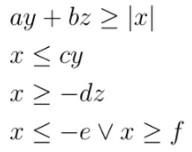
$$-\frac{1}{2}s_1-\frac{1}{2}s_3+\frac{1}{2}\leq 0$$

$$-\frac{1}{2}s_1-\frac{1}{2}s_3\leq-\frac{1}{2} \longrightarrow -\frac{1}{2}s_1-\frac{1}{2}s_3+s_4=-\frac{1}{2}$$

Adding slack for the cut eqn, adding it to the optimality table. Since Z row is optimal, and RHS contains a negative value, it fulfills the condition of being solved by dual simplex, where we get the pivot coln using min test. Until the RHS consists of only integer values, continue adding cutting constraints until feasible soln is found. **Limitations** – fractional cuts work with the assumption that all variables are integers, so RHS for slack vars must also be integers. And if x1 and x2 are fractional, slack must be fractional too, otherwise turn them all into integers.

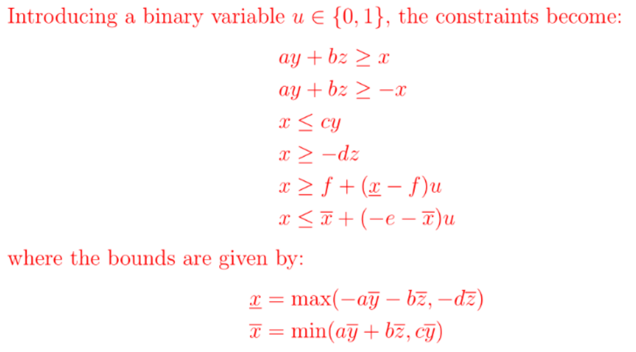
Gomory cuts are valid when the linear relaxation is bounded and produces a fractional solution

**NP-hard problems** are computational problems that are at least as difficult as the hardest problems in NP, meaning they likely require non-polynomial (often exponential) time to solve as the input size grows. Unlike NP problems, where solutions can be verified quickly, NP-hard problems may not even have a quick verification method. Why we use linear relaxation in MILPs: Faster Soln, Good L/U Bound for optimisation (LP solution will offer a stronger lower bound on the integer feasible optimal if it approximates the convex hull closely), approaching integer solns via cut, large problem feasibility. In a **branch-and-cut** algorithm, Gomory cuts are added iteratively to the LP relaxation as it is solved. Each cut narrows down the feasible region by removing fractional solutions, gradually steering the solution towards ILP convex hull. By combining branching and cutting, the branch-and-cut algorithm efficiently tightens the feasible region, improving the chances of finding the optimal integer solution faster.



$$ay + bz \geq |x|$$
$$x \leq cy$$
$$x \geq -dz$$
$$x \leq -e \lor x \geq f$$

Branching divides the parent node's feasible region into smaller parts, including its linear relaxation. The union of the child nodes' feasible regions is generally smaller, making their optimal solutions at least as large as the parent's optimal solution.

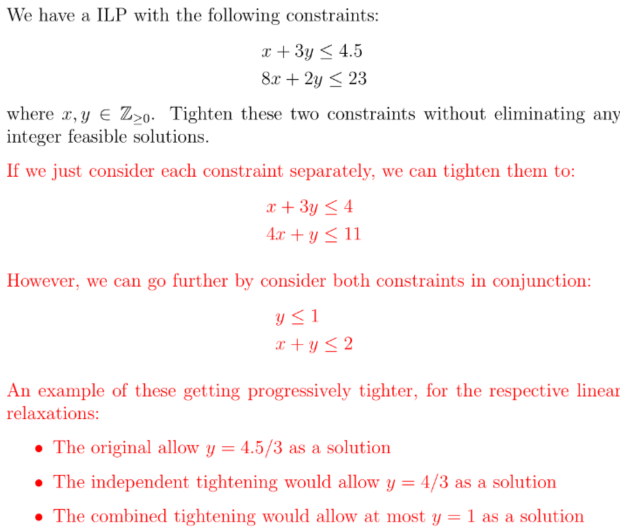


Introducing a binary variable $u \in \{0,1\}$, the constraints become:

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

where the bounds are given by:

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



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