Column Generation: LP with Complicating Constraints

Step-by-step Numerical Calculation + Code

Problem Data

Decision vector $x \in \mathbb{R}^8$ split into blocks:

$$B_1 = \{1, 2\},\ B_2 = \{3, 4\},\ B_3 = \{5, 6\},\ B_4 = \{7, 8\}.$$

Block sums $f = (5, 6, 4, 7)$, bounds $\bar{u} = (5, 5, 6, 6, 4, 4, 7, 7)$.

$$c = (3, 5, 4, 2, 1, 6, 2.5, 3.5), \quad A = \begin{bmatrix} 2 & 1 & 1 & 3 & 2 & 1 & 1 & 2 \\ 1 & 2 & 3 & 1 & 1 & 2 & 2 & 1 \\ 0 & 1 & 2 & 0 & 1 & 1 & 3 & 2 \end{bmatrix}, \quad b = (40, 32, 27).$$

Block equalities: $\sum_{j \in B_k} x_j = f_k$ for $k = 1, \ldots, 4$.

Algorithm Flow: Math $\leftrightarrow$ Code

Mathematical Steps:

1. Solve Master LP:

$$\min \sum u_s z_s + M \sum \delta_i$$

   Get duals: λ, σ

2. Solve Relaxed (pricing):

$$\bar{c} = c - A^\top \lambda$$

   For each block B_k:

$$\min_x \bar{c}^\top x \text{ s.t. } \sum x_j = f_k$$

Code Implementation:

```
while True:
  # Step 1: Master
  u, lam, sig, obj, delta
    = solve_master(
        Z_pool, R_pool, b)

  # Step 2: Pricing
  cbar = [c[j] - sum(
    lam[i]*A[i][j] ...)
    for j in range(n)]

  for k in range(blocks):
    x_blk = solve_block(
              k, cbar)

  v = sum(cbar[j]*x[j]
          ...)
```

Iteration Notation with Code Mapping

At iteration t:

$$\bar{c}^{(t)} = c - A^\top \lambda^{(t)}, \quad v^{(t)} = (\bar{c}^{(t)})^\top x^{(t)}, \quad z^{(t)} = c^\top x^{(t)}, \quad r^{(t)} = A x^{(t)}.$$

In code:

```
# Compute reduced costs: cbar = c - A^T * lambda
cbar = [c[j] - sum(lam[i]*A[i][j] for i in range(m))
        for j in range(n)]

# Solve each block subproblem with modified costs
for k in range(len(blocks)):
    x_blk = solve_block_modified_cost(k, cbar)  # min cbar*x s.
        t. sum(x)=f[k]

# Compute v = cbar^T * x, z = c^T * x, r = A*x
v = sum(cbar[j]*x[j] for j in range(n))
z = sum(c[j]*x[j] for j in range(n))
r = [sum(A[i][j]*x[j] for j in range(n)) for i in range(m)]
```

Convergence: $v^{(t)} \geq \sigma^{(t)}$ and $\delta \approx 0$.

Iteration 1: Duals and Reduced Costs

Step 1a: Get duals from Master LP:

$$\lambda^{(1)} = (10^6, -10^6, -10^6), \qquad \sigma^{(1)} = 4.70000605 \times 10^7.$$

Step 1b: Compute reduced costs $\bar{c}^{(1)} = c - A^\top \lambda^{(1)}$:

$$\bar{c}_1 = -9.99997 \times 10^5, \qquad \bar{c}_5 = 1,$$
$$\bar{c}_2 = 2.000005 \times 10^6, \qquad \bar{c}_6 = 2.000006 \times 10^6,$$
$$\bar{c}_3 = 4.000004 \times 10^6, \qquad \bar{c}_7 = 4.0000025 \times 10^6,$$
$$\bar{c}_4 = -1.999998 \times 10^6, \qquad \bar{c}_8 = 1.0000035 \times 10^6.$$

In code:

```
lam = [1e6, -1e6, -1e6]  # from Master dual
cbar = [c[j] - sum(lam[i]*A[i][j] for i in range(3)) for j in
    range(8)]
# cbar[0] = 3 - (2*1e6 + 1*(-1e6) + 0*(-1e6)) = -999997
```

Iteration 1: Block Choices and Values

Step 1c: Solve block subproblems (choose smallest $\bar{c}_j$):

- $B_1 = \{1, 2\}$: $\bar{c}_1 = -9.99997 \times 10^5 < \bar{c}_2 \Rightarrow x_1 = 5, x_2 = 0$
- $B_2 = \{3, 4\}$: $\bar{c}_4 = -1.999998 \times 10^6 < \bar{c}_3 \Rightarrow x_4 = 6, x_3 = 0$
- $B_3 = \{5, 6\}$: $\bar{c}_5 = 1 < \bar{c}_6 \Rightarrow x_5 = 4, x_6 = 0$
- $B_4 = \{7, 8\}$: $\bar{c}_8 = 1.0000035 \times 10^6 < \bar{c}_7 \Rightarrow x_8 = 7, x_7 = 0$

$$x^{(1)} = (5, 0, 0, 6, 4, 0, 0, 7)$$

```
# Block 1: indices [0,1]
x[0]=5.0, x[1]=0.0
# Block 2: indices [2,3]
x[2]=0.0, x[3]=6.0
```

$$v^{(1)} = -9.9999445 \times 10^6,$$
$$z^{(1)} = 55.5,$$
$$r^{(1)} = (50, 22, 18).$$

Since $v^{(1)} < \sigma^{(1)}$: add new column.

Iteration 2: Reduced Costs and Values

Iteration 2: $\lambda^{(2)} = (-0.263158, 10^6, -10^6)$, $\sigma^{(2)} = -3.999931 \times 10^6$

Reduced costs (picking smallest per block):

$$\begin{aligned}\bar{c}^{(2)} \approx (&-9.9999647\times10^5,\ -9.9999474\times10^5,\\ &-9.9999574\times10^5,\ -9.9999721\times10^5,\\ &1.526316,\ -9.9999374\times10^5,\\ &1.000002763\times10^6,\ 1.000004026\times10^6)\end{aligned}$$

Block choices: $x^{(2)} = (5, 0, 0, 6, 0, 4, 7, 0)$

Since $v^{(2)} < \sigma^{(2)}$: add column.

```
# B3: cbar[4]<cbar[5]
x[4]=4.0, x[5]=0.0
# B4: cbar[6]<cbar[7]
x[6]=7.0, x[7]=0.0

v = -7.999921e6
z = 68.5
r = (39, 33, 25)
```

Iteration 3: Reduced Costs and Values

Iteration 3: $\lambda^{(3)} = (0, 2.789474, -2.526316)$, $\sigma^{(3)} = 39.605263$

$$\bar{c}^{(3)} = (0.210526,\ 1.947368,\ 0.684210,\ -0.789474,\ 0.736842,\ 2.947368,\ 4.5,\ 5.763158)$$

Block choices:
B_1: $\bar{c}_1 = 0.21 < \bar{c}_2 \Rightarrow x_1 = 5$
B_2: $\bar{c}_4 = -0.79 < \bar{c}_3 \Rightarrow x_4 = 6$
B_3: $\bar{c}_5 = 0.74 < \bar{c}_6 \Rightarrow x_5 = 4$
B_4: $\bar{c}_7 = 4.5 < \bar{c}_8 \Rightarrow x_7 = 7$

Since $v^{(3)} < \sigma^{(3)}$: add column.

```
lam = [0, 2.789474,
       -2.526316]
cbar = [c[j] - sum(
  lam[i]*A[i][j]
  for i in range(3))
  for j in range(8)]

x = (5,0,0,6,4,0,7,0)
v = 30.763158
z = 48.5
r = (43, 29, 25)
```

Iteration 4: Convergence

Iteration 4: $\lambda^{(4)} = (0, 5, -4)$, $\sigma^{(4)} = 3.5$

$$\bar{c}^{(4)} = (-2, -1, -3, -3, 0, 0, 4.5, 6.5)$$

Block choices (tie in B_2):

$$x^{(4)} = (5, 0, 6, 0, 4, 0, 7, 0)$$

$$\begin{aligned} v^{(4)} &= (-2) \cdot 5 + (-3) \cdot 6 \\ &\quad + 0 \cdot 4 + 4.5 \cdot 7 \\ &= 3.5 = \sigma^{(4)} \end{aligned}$$

Converged! $v^{(4)} = \sigma^{(4)}$

```
lam = [0, 5, -4]
sig = 3.5

# Pricing:
cbar = ...  # computed
v = sum(cbar[j]*x[j]
        for j in range(8))

# Convergence check:
if v >= sig - 1e-9:
    print("Converged!")
    # Recover x* from
    # Master convex comb.
    x_star = sum(
      u[s]*X_pool[s]
      for s in range(P))
```

Recovered Optimal Solution and Check

Step 4: Recover $x^\star$ via master convex combination:

$$x^\star = \sum_{s=1}^{P} u_s^* \cdot X_{\text{pool}}[s]$$

Result:

$$x^\star = (5, 0, 1, 5, 3, 1, 7, 0)$$

$$z^\star = 55.5$$

Feasibility check:

$$Ax^\star = (40, 32, 27) = b,$$

$$\sum_{j \in B_k} x_j^\star = f_k,$$

$$0 \le x^\star \le \bar{u}.$$

```
# Master solution u_star
u_star = [u[s].X
          for s in range(P)]

# Convex combination
x_star = [0.0]*n
for s, us in enumerate(
                u_star):
    for j in range(n):
        x_star[j] += (
          us * X_pool[s][j])

z_star = sum(
  c[j]*x_star[j]
```

Iteration Summary

Iter	$\lambda^\top$	σ	v	z	$r^\top$
1	$(10^6, -10^6, -10^6)$	4.7×10^7	-9.9999×10^6	55.5	$(50, 22, 18)$
2	$(-0.26, 10^6, -10^6)$	-4.0×10^6	-7.9999×10^6	68.5	$(39, 33, 25)$
3	$(0, 2.789, -2.526)$	39.605	30.763	48.5	$(43, 29, 25)$
4	$(0, 5, -4)$	3.5	3.5	60.5	$(31, 41, 37)$

Code (1/4): Header & Data

```
# Column Generation for "LP with Complicating Constraints"
# Strictly: Initialization -> Master (always-feasible) ->
    Relaxed (block subproblems) -> convergence (v >= sigma) ->
    build x*
# Needs: gurobipy

from gurobipy import Model, GRB, quicksum

# ----------------------------
# Problem data (4 blocks, each 2 vars) & 3 complicating
    constraints
# ----------------------------
# x = (x1..x8)
c = [3.0, 5.0,   4.0, 2.0,   1.0, 6.0,   2.5, 3.5]    # cost
n = len(c)

# blocks: (x1,x2), (x3,x4), (x5,x6), (x7,x8)
blocks = [(0,2), (2,4), (4,6), (6,8)]
f = [5.0, 6.0, 4.0, 7.0]                                # per-
```

Code (2/4): Helpers & Initialization I

```
# ----------------------------
# Helpers
# ----------------------------
def solve_block_modified_cost(block_id, cbar):
    """ min sum cbar_j x_j  s.t. sum x_j = f[k], 0<=x_j<=ub_j "
        ""
    i0, i1 = blocks[block_id]
    M = Model()
    M.Params.OutputFlag = 0
    x = M.addVars(range(i0, i1), lb=0.0, ub=[ub[j] for j in
        range(i0, i1)],
                  name=f"x_blk{block_id}")
    M.addConstr(quicksum(x[j] for j in range(i0, i1)) == f[
        block_id])
    M.setObjective(quicksum(cbar[j]*x[j] for j in range(i0, i1)
        ), GRB.MINIMIZE)
```

Code (2/4): Helpers & Initialization II

```
    M.optimize()
    x_blk = [0.0]*n
    for j in range(i0, i1):
        x_blk[j] = x[j].X
    return x_blk

def eval_obj(x):
    return sum(c[j]*x[j] for j in range(n))

def eval_r(x):
    return [sum(A[i][j]*x[j] for j in range(n)) for i in range(
        m)]

# ----------------------------
# Initialization: deliberately few columns
# ----------------------------
init_costs = [
```

Code (2/4): Helpers & Initialization III

```
    [-1, -1,  1,  2,  3,  4,  3,  4],  # seed Block1
    [ 2,  1, -1, -1,  3,  4,  3,  4],  # seed Block2
    [ 3,  3,  2,  3, -1, -1,  2,  3],  # seed Block3
    # (leave Block4 to be generated by pricing)
]

X_pool, Z_pool, R_pool = [], [], []
for hat_c in init_costs:
    x_total = [0.0]*n
    for k in range(len(blocks)):
        x_blk = solve_block_modified_cost(k, hat_c)
        for j in range(n):
            x_total[j] += x_blk[j]
    X_pool.append(x_total)
    Z_pool.append(eval_obj(x_total))
    R_pool.append(eval_r(x_total))
```

Code (2/4): Helpers & Initialization IV

```
print(f"Initialized with {len(X_pool)} columns.")
```

Code (3/4): Always-feasible Master & Relaxed I

```
# ----------------------------
# Always-feasible Master
# ----------------------------
def solve_master(Z_pool, R_pool, b, BIGM=1e6):
    P = len(Z_pool)
    M = Model()
    M.Params.OutputFlag = 0
    u = M.addVars(P, lb=0.0, name="u")
    dpos = M.addVars(m, lb=0.0, name="dpos")
    dneg = M.addVars(m, lb=0.0, name="dneg")

    M.setObjective(quicksum(Z_pool[s]*u[s] for s in range(P))
                   + BIGM*quicksum(dpos[i]+dneg[i] for i in
                       range(m)), GRB.MINIMIZE)

    cc = []
```

Code (3/4): Always-feasible Master & Relaxed II

```
    for i in range(m):
        cons = M.addConstr(quicksum(R_pool[s][i]*u[s] for s in
            range(P))
                           + dpos[i] - dneg[i] == b[i], name=f"
                               Aeq[{i}]")
        cc.append(cons)

    conv = M.addConstr(quicksum(u[s] for s in range(P)) == 1.0,
        name="convexity")
    M.optimize()

    if M.Status != GRB.OPTIMAL:
        raise RuntimeError(f"Master not optimal, status={M.
            Status}")

    u_star = [u[s].X for s in range(P)]
    lam = [cc[i].Pi for i in range(m)]
```

Code (3/4): Always-feasible Master & Relaxed III

```
    sig = conv.Pi
    obj = M.ObjVal
    deltas = ([dpos[i].X for i in range(m)], [dneg[i].X for i
        in range(m)])
    return u_star, lam, sig, obj, deltas

# ----------------------------
# Relaxed/pricing: cbar = c - A^T lambda
# ----------------------------
def solve_relaxed(lam):
    cbar = [c[j] - sum(lam[i]*A[i][j] for i in range(m)) for j
        in range(n)]
    x_total = [0.0]*n
    for k in range(len(blocks)):
        x_blk = solve_block_modified_cost(k, cbar)
        for j in range(n):
            x_total[j] += x_blk[j]
```

Code (3/4): Always-feasible Master & Relaxed IV

```
    v = sum(cbar[j]*x_total[j] for j in range(n))
    z = eval_obj(x_total)
    r = eval_r(x_total)
    return x_total, v, z, r, cbar
```

Code (4/4): CG Loop & Full LP Compare I

```
# ----------------------------
# Column Generation Loop
# ----------------------------
it = 0
while True:
    it += 1
    u_star, lam, sig, obj, deltas = solve_master(Z_pool, R_pool
        , b)
    x_new, v, z_new, r_new, cbar = solve_relaxed(lam)
    dpos, dneg = deltas
    max_dev = max([0.0] + dpos + dneg)

    print(f"\n=== Iteration {it} ===")
    print(f"Master obj: {obj:.6f}")
    print(f"Duals \u03BB: {', '.join(f'{x:.6f}' for x in lam)}
        \u03C3: {sig:.6f}")
```

Code (4/4): CG Loop & Full LP Compare II

```
    print(f"Relaxed v: {v:.6f}   \u03C3: {sig:.6f}   max|\u03B4
        |: {max_dev:.2e}")

    if (v >= sig - 1e-9) and (max_dev <= 1e-7):
        x_star = [0.0]*n
        for s, us in enumerate(u_star):
            for j in range(n):
                x_star[j] += us * X_pool[s][j]
        z_star = eval_obj(x_star)
        r_star = eval_r(x_star)
        print("\nConverged: v >= \u03C3 and \u03B40.")
        print("x* =", [round(val,6) for val in x_star])
        print("z* =", round(z_star,6))
        print("A x* =", [round(val,6) for val in r_star], " (==
            b)")
        break
```

Code (4/4): CG Loop & Full LP Compare III

```
    X_pool.append(x_new)
    Z_pool.append(z_new)
    R_pool.append(r_new)
    print(">> Added new column and continue...")

# ----------------------------
# Direct full LP for verification
# ----------------------------
def solve_full_lp():
    M = Model("full_lp")
    M.Params.OutputFlag = 0
    x = M.addVars(range(n), lb=0.0, ub=ub, name="x")
    for k, (i0, i1) in enumerate(blocks):
        M.addConstr(quicksum(x[j] for j in range(i0, i1)) == f[
            k], name=f"blk[{k}]")
    for i in range(m):
```

Code (4/4): CG Loop & Full LP Compare IV

```
        M.addConstr(quicksum(A[i][j] * x[j] for j in range(n))
            == b[i], name=f"Aeq[{i}]")
    M.setObjective(quicksum(c[j] * x[j] for j in range(n)), GRB
        .MINIMIZE)
    M.optimize()
    if M.Status != GRB.OPTIMAL:
        raise RuntimeError(f"Full LP not optimal, status={M.
            Status}")
    x_full = [x[j].X for j in range(n)]
    z_full = sum(c[j] * x_full[j] for j in range(n))
    return x_full, z_full

x_full, z_full = solve_full_lp()
print("\n--- Direct (one-shot) LP solve ---")
print("x_full =", [round(v, 6) for v in x_full])
print("z_full =", round(z_full, 6))
```