Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Algorithmik

Algorithm Engineering
Lecture 11: Decomposition

Problem Decomposition: Why?

Problem Decomposition: Why?

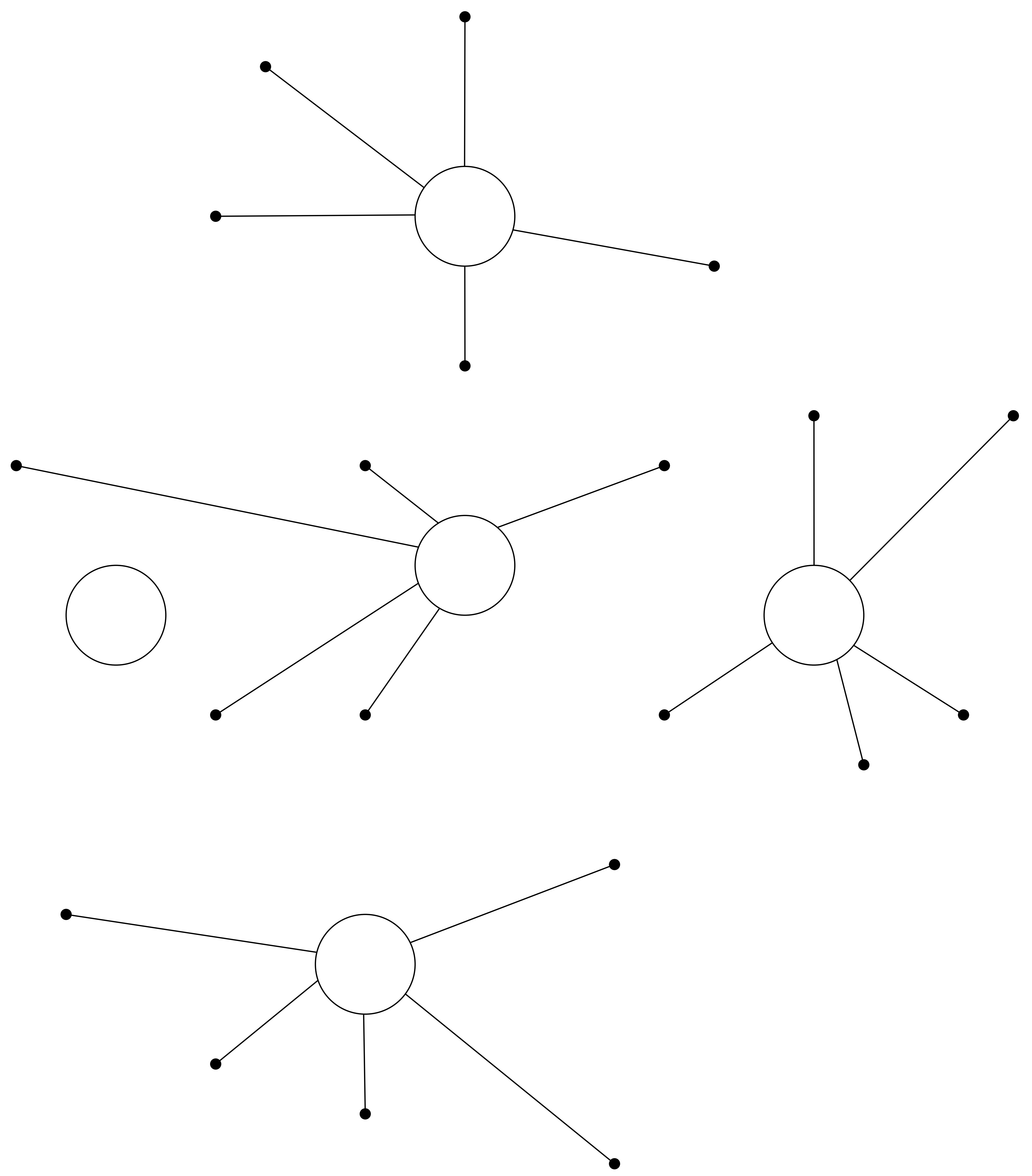
Some hard optimization problems:

- Composition of relatively loosely coupled individual problems
- Some components may already be hard
- **Or:** local decisions combined by a few global constraints
- **Idea:** Use that structure (*decompose* the problem)
- **This lecture:** how can we do that?

Example: Facility Location

Given:

- potential facility locations i
  - opening cost f_i
  - capacities c_i
- n clients j
  - every client must be assigned a facility
  - service costs s_{ij}
- at most c_i clients at facility i
- objective: minimize costs



Example: Facility Location — Single MIP

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

$\forall$facilities i: $\sum_j y_{ij} \leq c_i x_i$

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i\text{:} \quad \sum_j y_{ij} \le c_i x_i$$

$$\forall \text{facilities } i, \text{ clients } j : y_{ij} \le x_i$$

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i\text{:} \sum_j y_{ij} \leq c_i x_i$$

$$\forall \text{facilities } i, \text{ clients } j : y_{ij} \leq x_i$$

$$\forall \text{clients } j : \sum_i y_{ij} = 1$$

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i\text{: } \sum_j y_{ij} \leq c_i x_i$$

$$\forall \text{facilities } i\text{, clients } j : y_{ij} \leq x_i$$

$$\forall \text{clients } j : \sum_i y_{ij} = 1$$

$$x_i, y_{ij} \in \{0,1\}$$

Example: Facility Location — Single MIP

Variables: x_i (opening facility i), y_{ij} (client assignment)

$$\min \sum_i f_i x_i + \sum_j \sum_i s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i\text{:} \quad \sum_j y_{ij} \leq c_i x_i$$

$$\forall \text{facilities } i, \text{ clients } j : y_{ij} \leq x_i$$

$$\forall \text{clients } j : \sum_i y_{ij} = 1$$

$$x_i, y_{ij} \in \{0,1\}$$

Now: three decomposition methods, three different decompositions.

First Decomposition: Lagrangian Relaxation

Lagrangian relaxation: constraints → objective penalties

First Decomposition: Lagrangian Relaxation

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j : \sum_i a_{ij} x_i = b_j \rightarrow \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

First Decomposition: Lagrangian Relaxation

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j : \sum_i a_{ij} x_i = b_j \rightarrow \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

Why is this called a relaxation? Any original solution is still there with the same objective!

First Decomposition: Lagrangian Relaxation

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j : \sum_i a_{ij} x_i = b_j \to \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

Why is this called a relaxation? Any original solution is still there with the same objective!

After solving with fixed penalties μ_j, update the penalties and iterate.

First Decomposition: Lagrangian Relaxation

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j: \sum_i a_{ij} x_i = b_j \rightarrow \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

Why is this called a relaxation? Any original solution is still there with the same objective!

After solving with fixed penalties μ_j, update the penalties and iterate.

Allows us to remove a coupling subset of the constraints!

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j : \sum_i a_{ij} x_i = b_j \rightarrow \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

Why is this called a relaxation? Any original solution is still there with the same objective!

After solving with fixed penalties μ_j, update the penalties and iterate.

Allows us to remove a coupling subset of the constraints!

Eliminate coupling constraints:

Lagrangian relaxation: constraints → objective penalties

Idea in general: $\min c_i x_i$ s.t. $\forall j : \sum_i a_{ij} x_i = b_j \rightarrow \min c_i x_i + \sum_j \mu_j (b_j - \sum_i a_{ij} x_i)$

Why is this called a relaxation? Any original solution is still there with the same objective!

After solving with fixed penalties μ_j, update the penalties and iterate.

Allows us to remove a coupling subset of the constraints!

Eliminate coupling constraints:

In our case: $\forall$clients $j : \sum_i y_{ij} = 1$; without these, we can solve every facility on its own.

Decomposable Lagrangian Relaxation on Facility Location

Decomposable Lagrangian Relaxation on Facility Location

Here: $\forall$clients j : $\sum_i y_{ij} = 1 \rightarrow \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Decomposable Lagrangian Relaxation on Facility Location

Here: $\forall$clients $j : \sum_i y_{ij} = 1 \rightarrow \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Rewritten: $\min \sum_i x_i f_i + \sum_j \sum_i (s_{ij} - \mu_j) y_{ij} + \sum_j \mu_j$ s.t.

Here: $\forall$clients j : $\sum_i y_{ij} = 1 \rightarrow \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Rewritten: $\min \sum_i x_i f_i + \sum_j \sum_i (s_{ij} - \mu_j) y_{ij} + \sum_j \mu_j$ s.t.

$\forall$facilities i: $\sum_j y_{ij} \leq c_i x_i$

Decomposable Lagrangian Relaxation on Facility Location

Here: $\forall$clients j : $\sum_i y_{ij} = 1 \to \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Rewritten: $\min \sum_i x_i f_i + \sum_j \sum_i (s_{ij} - \mu_j) y_{ij} + \sum_j \mu_j$ s.t.

$\forall$facilities i: $\sum_j y_{ij} \le c_i x_i$

$\forall$facilities i, clients j : $y_{ij} \le x_i$

Decomposable Lagrangian Relaxation on Facility Location

Here: $\forall$clients j : $\sum_i y_{ij} = 1 \rightarrow \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Rewritten: $\min \sum_i x_i f_i + \sum_j \sum_i (s_{ij} - \mu_j) y_{ij} + \sum_j \mu_j$ s.t.

$\forall$facilities i: $\sum_j y_{ij} \leq c_i x_i$

$\forall$facilities i, clients j : $y_{ij} \leq x_i$

$x_i, y_{ij} \in \{0,1\}$

Decomposable Lagrangian Relaxation on Facility Location

Here: $\forall$clients j : $\sum_i y_{ij} = 1 \rightarrow \min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Rewritten: $\min \sum_i x_i f_i + \sum_j \sum_i (s_{ij} - \mu_j) y_{ij} + \sum_j \mu_j$ s.t.

$\forall$facilities i: $\sum_j y_{ij} \leq c_i x_i$

$\forall$facilities i, clients j : $y_{ij} \leq x_i$

$x_i, y_{ij} \in \{0,1\}$

Independent subproblem for each facility!

Decomposable Lagrangian Relaxation: Subproblems

Each facility i:

Each facility i:

$$\min x_i f_i + \sum_j (s_{ij} - \mu_j) y_{ij} \text{ s.t.}$$

Each facility i:

$$\min x_i f_i + \sum_j (s_{ij} - \mu_j) y_{ij} \text{ s.t.}$$

$$\sum_j y_{ij} \leq c_i x_i$$

Each facility i:

$$\min x_i f_i + \sum_j (s_{ij} - \mu_j) y_{ij} \text{ s.t.}$$

$$\sum_j y_{ij} \le c_i x_i$$

$$\forall \text{clients } j : y_{ij} \le x_i$$

Each facility i:

$$\min x_i f_i + \sum_j (s_{ij} - \mu_j) y_{ij} \text{ s.t.}$$

$$\sum_j y_{ij} \leq c_i x_i$$

$\forall$clients $j : y_{ij} \leq x_i$

$$x_i, y_{ij} \in \{0,1\}$$

Decomposable Lagrangian Relaxation: Subproblems

Each facility i:

$$\min x_i f_i + \sum_j (s_{ij} - \mu_j) y_{ij} \text{ s.t.}$$

$$\sum_j y_{ij} \leq c_i x_i$$

$$\forall \text{clients } j : y_{ij} \leq x_i$$

$$x_i, y_{ij} \in \{0,1\}$$

Very efficiently solvable in practice; how?

Decomposable Lagrangian Relaxation: Subproblems

Result of subproblem: $L(\mu)$

We know: $L(\mu) \leq \mathrm{OPT}$ (for minimization; we are solving a relaxation!)

Result of subproblem: $L(\mu)$

We know: $L(\mu) \leq \mathrm{OPT}$ (for minimization; we are solving a relaxation!)

Only equality constraints: This holds for any vector μ!

Decomposable Lagrangian Relaxation: Subproblems

Result of subproblem: $L(\mu)$

We know: $L(\mu) \leq \mathrm{OPT}$ (for minimization; we are solving a relaxation!)

Only equality constraints: This holds for any vector μ!

For inequality constraints: bound to $\mu_j \geq 0$ or $\mu_j \leq 0$ (in such a way that a 'slack' constraint cannot be penalized by a multiplier/penalty μ_j with the wrong sign).

Result of subproblem: $L(\mu)$

We know: $L(\mu) \leq \mathrm{OPT}$ (for minimization; we are solving a relaxation!)

Only equality constraints: This holds for any vector μ!

For inequality constraints: bound to $\mu_j \geq 0$ or $\mu_j \leq 0$ (in such a way that a 'slack' constraint cannot be penalized by a multiplier/penalty μ_j with the wrong sign).

Otherwise, you can get wrong results (i.e., $L(\mu) > \mathrm{OPT}$ can happen; there need not be a solution in which all inequalities are tight, thus penalized slack is bad!)

Result of subproblem: $L(\mu)$

We know: $L(\mu) \leq \mathrm{OPT}$ (for minimization; we are solving a relaxation!)

Only equality constraints: This holds for any vector μ!

For inequality constraints: bound to $\mu_j \geq 0$ or $\mu_j \leq 0$ (in such a way that a 'slack' constraint cannot be penalized by a multiplier/penalty μ_j with the wrong sign).

Otherwise, you can get wrong results (i.e., $L(\mu) > \mathrm{OPT}$ can happen; there need not be a solution in which all inequalities are tight, thus penalized slack is bad!)

Interpretation of μ: prices for violating constraints; we aim to iteratively find μ to maximize our bound $L(\mu)$, essentially solving a *Lagrangian dual* problem.

Decomposable Lagrangian Relaxation: Loop

So the loop becomes:

- Initialize penalties μ (e.g., to 0 or the LP dual values of the on the full problem)
- Solve the subproblems for the fixed μ; result:
  - This gives a lower bound (optimal solution of the relaxation)
  - Make heuristic solution from relaxed solution (for an upper bound)
- Update the penalties μ (cut-off/projection to ensure non-negativity/non-positivity)
- If lower bound $\neq$ upper bound, repeat.
- **In general:** needs branching for completeness or some termination criterion.

Decomposable Lagrangian Relaxation: Updating Penalties

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $$\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$$

- Negative μ_j: penalizes over-covering a client

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $$\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$$

- Negative μ_j: penalizes over-covering a client
- Positive μ_j: penalizes under-covering a client

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

- Negative μ_j: penalizes over-covering a client
- Positive μ_j: penalizes under-covering a client
- Let $n_j = \sum_i y_{ij}$ be the coverage count of client j, $g_j = 1 - n_j$, in general $g =$ violations

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

- Negative μ_j: penalizes over-covering a client
- Positive μ_j: penalizes under-covering a client
- Let $n_j = \sum_i y_{ij}$ be the coverage count of client j, $g_j = 1 - n_j$, in general $g =$ violations
- **Updating rule:** for example, $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step

Decomposable Lagrangian Relaxation: Updating Penalties

Penalties: $\min \sum_i x_i f_i + \sum_j \sum_i s_{ij} y_{ij} + \sum_j \mu_j \left(1 - \sum_i y_{ij}\right)$

- Negative μ_j: penalizes over-covering a client
- Positive μ_j: penalizes under-covering a client
- Let $n_j = \sum_i y_{ij}$ be the coverage count of client j, $g_j = 1 - n_j$, in general $g =$ violations
- **Updating rule:** for example, $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step
- This updating rule is also called *subgradient method*

Details — Decomposable Lagrangian Relaxation: Step Lengths

Details — Decomposable Lagrangian Relaxation: Step Lengths

Subgradient method update: $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step

Details — Decomposable Lagrangian Relaxation: Step Lengths

Subgradient method update: $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step

Diminishing & non-summable: $\lim_{k\to\infty} t_k = 0, \sum_k t_k = \infty$ converges, can be slow

Details — Decomposable Lagrangian Relaxation: Step Lengths

Subgradient method update: $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step

Diminishing & non-summable: $\lim_{k\to\infty} t_k = 0,\ \sum_k t_k = \infty$ converges, can be slow

Polyak/Held-Karp: $t_k = \lambda_k \frac{UB - L(\mu^k)}{||g^k||_2^2}$, starting with $\lambda_k = 2$ and halving λ after some p iterations without bound improvement; convergence needs $\lambda_k \in (0,2)$.
Sometimes, λ_k is reset on finding a better incumbent/UB.

Details — Decomposable Lagrangian Relaxation: Step Lengths

Subgradient method update: $\mu_j \leftarrow \mu_j + t_k g_j$ for some step length t_k in the kth step

Diminishing & non-summable: $\lim_{k\to\infty} t_k = 0,\ \sum_k t_k = \infty$ converges, can be slow

Polyak/Held-Karp: $t_k = \lambda_k \frac{UB - L(\mu^k)}{||g^k||_2^2}$, starting with $\lambda_k = 2$ and halving λ after some p iterations without bound improvement; convergence needs $\lambda_k \in (0,2)$.
Sometimes, λ_k is reset on finding a better incumbent/UB.

Alternative methods: deflected subgradient (also add in previous step direction), …

Alternative Decomposition: Dantzig-Wolfe

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

For each facility & pattern: C_i^p (clients served by i in pattern p), s_i^p (cost of pattern p)

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

For each facility & pattern: C_i^p (clients served by i in pattern p), s_i^p (cost of pattern p)

Master problem: $\min \sum_{i} \sum_{p \in P_i} x_i^p s_i^p$ s.t.

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

For each facility & pattern: C_i^p (clients served by i in pattern p), s_i^p (cost of pattern p)

Master problem: $\min \sum_{i} \sum_{p \in P_i} x_i^p s_i^p$ s.t.

$$\forall \text{facilities } i : \sum_{p \in P_i} x_i^p = 1, \qquad \forall \text{clients } j : \sum_{j} \sum_{p, j \in C_i^p} x_i^p = 1$$

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

For each facility & pattern: C_i^p (clients served by i in pattern p), s_i^p (cost of pattern p)

Master problem: $\min \sum_i \sum_{p \in P_i} x_i^p s_i^p$ s.t.

$$\forall \text{facilities } i: \sum_{p \in P_i} x_i^p = 1, \qquad \forall \text{clients } j: \sum_j \sum_{p, j \in C_i^p} x_i^p = 1$$

$$x_i^p \geq 0$$

Alternative Decomposition: Dantzig-Wolfe

Basic observation: solutions for individual facilities can be combined.

Idea: each facility makes a locally feasible choice; a master problem coordinates these

Choices (patterns) for each facility: which clients are served (if none, facility closed)

For each facility i: P_i (feasible patterns)

For each facility & pattern: C_i^p (clients served by i in pattern p), s_i^p (cost of pattern p)

Master problem: $\min \sum_{i} \sum_{p \in P_i} x_i^p s_i^p$ s.t.

$$\forall \text{facilities } i: \sum_{p \in P_i} x_i^p = 1, \qquad \forall \text{clients } j: \sum_{j} \sum_{p, j \in C_i^p} x_i^p = 1$$

$$x_i^p \geq 0$$

Each locally feasible choice is a variable now!

Column Generation

Obvious problem: way too many variables!

Obvious problem: way too many variables!

Solution: column generation!

Obvious problem: way too many variables!

Solution: column generation!

Idea: start with a subset, then generate & add variables lazily.

Obvious problem: way too many variables!

Solution: column generation!

Idea: start with a subset, then generate & add variables lazily.

Simplex switches one variable with *negative reduced costs* into its basis at a time.

Column Generation

Obvious problem: way too many variables!

Solution: column generation!

Idea: start with a subset, then generate & add variables lazily.

Simplex switches one variable with *negative reduced costs* into its basis at a time.

Such variables correspond to violated constraints in the dual LP;
if there is no such variable, the solution is optimal.

Obvious problem: way too many variables!

Solution: column generation!

Idea: start with a subset, then generate & add variables lazily.

Simplex switches one variable with *negative reduced costs* into its basis at a time.

Such variables correspond to violated constraints in the dual LP;
if there is no such variable, the solution is optimal.

Refined idea: at the LP optimum of the restricted master problem (RMP), find the variable minimizing the reduced costs. If it has negative reduced costs, add it; otherwise, the LP solution is optimal.

Obvious problem: way too many variables!

Solution: column generation!

Idea: start with a subset, then generate & add variables lazily.

Simplex switches one variable with *negative reduced costs* into its basis at a time.

Such variables correspond to violated constraints in the dual LP;
if there is no such variable, the solution is optimal.

Refined idea: at the LP optimum of the restricted master problem (RMP), find the variable minimizing the reduced costs. If it has negative reduced costs, add it; otherwise, the LP solution is optimal.

Finding this variable is typically done via solving a *pricing* (sub-)problem.

Column Generation for Facility Location: Dual of the Master

Column Generation for Facility Location: Dual of the Master

Master problem: $\min \sum_i \sum_{p \in P_i} s_i^p$ s.t.

$\forall$facilities i : $\sum_{p \in P_i} x_i^p = 1$, $\forall$clients j : $\sum_j \sum_{p, j \in C_i^p} x_i^p = 1$

$$x_i^p \geq 0$$

Master problem: $\min \sum_{i} \sum_{p \in P_i} s_i^p$ s.t.

$\forall$facilities i : $\sum_{p \in P_i} x_i^p = 1$, $\quad$ $\forall$clients j : $\sum_{j} \sum_{p, j \in C_i^p} x_i^p = 1$

$$x_i^p \geq 0$$

Dual: $\max \sum_{i} \alpha_i + \sum_{j} \beta_j$ s.t.

Master problem: $\min \sum_i \sum_{p \in P_i} s_i^p$ s.t.

$\forall$facilities i : $\sum_{p \in P_i} x_i^p = 1$, $\forall$clients j : $\sum_j \sum_{p, j \in C_i^p} x_i^p = 1$

$x_i^p \geq 0$

Dual: $\max \sum_i \alpha_i + \sum_j \beta_j$ s.t.

$\forall$facilities i, patterns $p \in P_i$: $\alpha_i + \sum_{j \in p} \beta_j \leq s_i^p$; reduced costs: $s_i^p - \alpha_i - \sum_{j \in p} \beta_j$

Reduced costs: $s_i^p - \alpha_i - \sum_{j \in p} \beta_j$; we get α_i, β_j from solving the RMP to optimality.

Pricing problem for a fixed facility i, assuming disabled facility is already in the RMP:

$\min f_i - \alpha_i + \sum_j (s_{ij} - \beta_j) y_j$ s.t.

$$\sum_j y_j \leq c_i,$$

$$y_j \in \{0,1\}$$

Very easy to solve (greedy works!); essentially the same that we saw for Lagrange!

Details — Column Generation: Pricing and Stabilization

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Early iterations: meaningless, often later useless columns

Details — Column Generation: Pricing and Stabilization

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Early iterations: meaningless, often later useless columns

Late iterations: jumping around between almost-optimal solutions

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Early iterations: meaningless, often later useless columns

Late iterations: jumping around between almost-optimal solutions

Two fixes: *stabilization* and *stopping rules*

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Early iterations: meaningless, often later useless columns

Late iterations: jumping around between almost-optimal solutions

Two fixes: *stabilization* and *stopping rules*

Stabilization: add slack variables in the RMP (allow violating coupling constraints at a cost); often piecewise-linear penalties. Prevents the dual from jumping too much from a *stability center* (a dual vector encoded in the penalties); only move the center on sufficient bound improvement.

Details — Column Generation: Pricing and Stabilization

Practical issue: We solve the (dual of) the RMP in each iteration, adding constraints. Its solution can jump wildly and be far off the true dual optimum.

Early iterations: meaningless, often later useless columns

Late iterations: jumping around between almost-optimal solutions

Two fixes: *stabilization* and *stopping rules*

Stabilization: add slack variables in the RMP (allow violating coupling constraints at a cost); often piecewise-linear penalties. Prevents the dual from jumping too much from a *stability center* (a dual vector encoded in the penalties); only move the center on sufficient bound improvement.

Stopping rules: compute a lower bound on the LP (often get this from pricing for free).

Details — Branch & Price

Combining column generation and branching:

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)
- We then run branch & bound solving the RMP via Column Generation in each node

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)
- We then run branch & bound solving the RMP via Column Generation in each node
- We get UB’s and LB’s on the LP; we can stop early if we want (or can prune)

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)
- We then run branch & bound solving the RMP via Column Generation in each node
- We get UB's and LB's on the LP; we can stop early if we want (or can prune)
- We occasionally have to generate columns to escape infeasibility; usually a Farkas' lemma infeasibility certificate (dual ray) is used and the same pricing problem is solved as usual, but with different objective coefficients (no details here)

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)
- We then run branch & bound solving the RMP via Column Generation in each node
- We get UB's and LB's on the LP; we can stop early if we want (or can prune)
- We occasionally have to generate columns to escape infeasibility;
  usually a Farkas' lemma infeasibility certificate (dual ray) is used and the same pricing problem is solved as usual, but with different objective coefficients (no details here)
- Reasonable cheap heuristics (e.g., rounding heuristics) can get solutions from the RMP

Combining column generation and branching:

- The RMP is solved as an LP and gives a bound (stronger than the original LP)
- We then run branch & bound solving the RMP via Column Generation in each node
- We get UB's and LB's on the LP; we can stop early if we want (or can prune)
- We occasionally have to generate columns to escape infeasibility;
  usually a Farkas' lemma infeasibility certificate (dual ray) is used and the same pricing problem is solved as usual, but with different objective coefficients (no details here)
- Reasonable cheap heuristics (e.g., rounding heuristics) can get solutions from the RMP
- Column generation can be used outside of decomposition purposes (e.g., TSP)

Details — Branch & Price

Combining column generation and branching:

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)
- Branching on master problem variables is awkward; branch on original variables instead

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)
- Branching on master problem variables is awkward; branch on original variables instead
- E.g., branching down one x_i^p forbids it in the pricing problem; hard to implement

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)
- Branching on master problem variables is awkward; branch on original variables instead
- E.g., branching down one x_i^p forbids it in the pricing problem; hard to implement
- Instead, branching on x_i or y_{ij} (original variables) works in the pricing problem; in the master, it translates to expressible constraints

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)
- Branching on master problem variables is awkward; branch on original variables instead
- E.g., branching down one x_i^p forbids it in the pricing problem; hard to implement
- Instead, branching on x_i or y_{ij} (original variables) works in the pricing problem; in the master, it translates to expressible constraints
- Cutting planes can be difficult: some affect the dual and must be considered in pricing; cuts that can be expressed linearly on the original variables are typically fine

Combining column generation and branching:

- Not as straightforward as it may seem; not supported by Gurobi, CPLEX & Co.
- Supported in SCIP (which can use, e.g., Gurobi as a backend to solve individual LPs)
- Branching on master problem variables is awkward; branch on original variables instead
- E.g., branching down one x_i^p forbids it in the pricing problem; hard to implement
- Instead, branching on x_i or y_{ij} (original variables) works in the pricing problem; in the master, it translates to expressible constraints
- Cutting planes can be difficult: some affect the dual and must be considered in pricing; cuts that can be expressed linearly on the original variables are typically fine
- Effective use is considered an art by some mathematical programming researchers

Third Decomposition Approach: Benders' Decomposition

Approaches so far:

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate

Third Decomposition Approach: Benders' Decomposition

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate
- Enumerate promising solutions to those 'easy' subproblems, solve coordinating master

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate
- Enumerate promising solutions to those 'easy' subproblems, solve coordinating master
- **Next:** Benders' Decomposition

Third Decomposition Approach: Benders' Decomposition

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate
- Enumerate promising solutions to those 'easy' subproblems, solve coordinating master
- **Next:** Benders' Decomposition
  - Extract some 'hard' part into a master problem (e.g., which facilities to open)

Third Decomposition Approach: Benders' Decomposition

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate
- Enumerate promising solutions to those 'easy' subproblems, solve coordinating master
- **Next:** Benders' Decomposition
  - Extract some 'hard' part into a master problem (e.g., which facilities to open)
  - Use a relaxation to estimate the costs of the overall solution

Third Decomposition Approach: Benders' Decomposition

Approaches so far:

- Dualize away coupling constraints, solve 'easy' subproblem for each facility, iterate
- Enumerate promising solutions to those 'easy' subproblems, solve coordinating master
- **Next:** Benders' Decomposition
  - Extract some 'hard' part into a master problem (e.g., which facilities to open)
  - Use a relaxation to estimate the costs of the overall solution
  - Add cutting planes from a subproblem to refine the cost estimate
    (e.g., what is the best value for covering all clients with a given set of open facilities)

Third Decomposition Approach: Benders' Decomposition

Concrete master:

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t. } \quad \text{(cost estimate + correction term } \theta \text{ ('epigraph variable'))}$$

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t.}$$ (cost estimate + correction term θ ('epigraph variable'))

$$\sum_i c_i x_i \geq n =$$ #clients (relaxation of the subproblem (feasibility))

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t.}$$ (cost estimate + correction term θ ('epigraph variable'))

$$\sum_i c_i x_i \geq n = \text{\#clients}$$ (relaxation of the subproblem (feasibility))

+ growing set of Benders' feasibility & optimality cuts

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t.}$$ (cost estimate + correction term θ ('epigraph variable'))

$$\sum_i c_i x_i \geq n =$$ #clients (relaxation of the subproblem (feasibility))

+ growing set of Benders' feasibility & optimality cuts

$$x_i \in \{0,1\}, \theta \geq 0$$

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t. }$$ (cost estimate + correction term θ ('epigraph variable'))

$$\sum_i c_i x_i \geq n =$$ #clients (relaxation of the subproblem (feasibility))

+ growing set of Benders' feasibility & optimality cuts

$$x_i \in \{0,1\}, \theta \geq 0$$

Each solution of the master: solution (of the 'hard' part) + optimistic cost estimate

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t.} \quad \text{(cost estimate + correction term } \theta \text{ ('epigraph variable'))}$$

$$\sum_i c_i x_i \geq n = \#\text{clients} \quad \text{(relaxation of the subproblem (feasibility))}$$

+ growing set of Benders' feasibility & optimality cuts

$$x_i \in \{0,1\}, \theta \geq 0$$

Each solution of the master: solution (of the 'hard' part) + optimistic cost estimate

Subproblem: compute feasibility & actual costs (actual solution) of that partial solution

Concrete master:

$$\min \sum_i f_i x_i + \theta \text{ s.t. } \quad \text{(cost estimate + correction term } \theta \text{ ('epigraph variable'))}$$

$$\sum_i c_i x_i \geq n = \text{\#clients} \quad \text{(relaxation of the subproblem (feasibility))}$$

+ growing set of Benders' feasibility & optimality cuts

$$x_i \in \{0,1\}, \theta \geq 0$$

Each solution of the master: solution (of the 'hard' part) + optimistic cost estimate

Subproblem: compute feasibility & actual costs (actual solution) of that partial solution

So: given a fixed set of open facilities, optimally solve the client assignment problem

Third Decomposition Approach: Benders' Decomposition

Concrete subproblem:

Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_{j} s_{ij} y_{ij} \text{ s.t.}$$

Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_j s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i: \sum_j y_{ij} \leq x_i c_i$$

Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_j s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i : \sum_j y_{ij} \leq x_i c_i$$

$$\forall \text{clients } j : \sum_i y_{ij} = 1$$

Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_j s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i : \sum_j y_{ij} \le x_i c_i$$

$$\forall \text{clients } j : \sum_i y_{ij} = 1$$

$$y_{ij} \in \{0,1\}$$

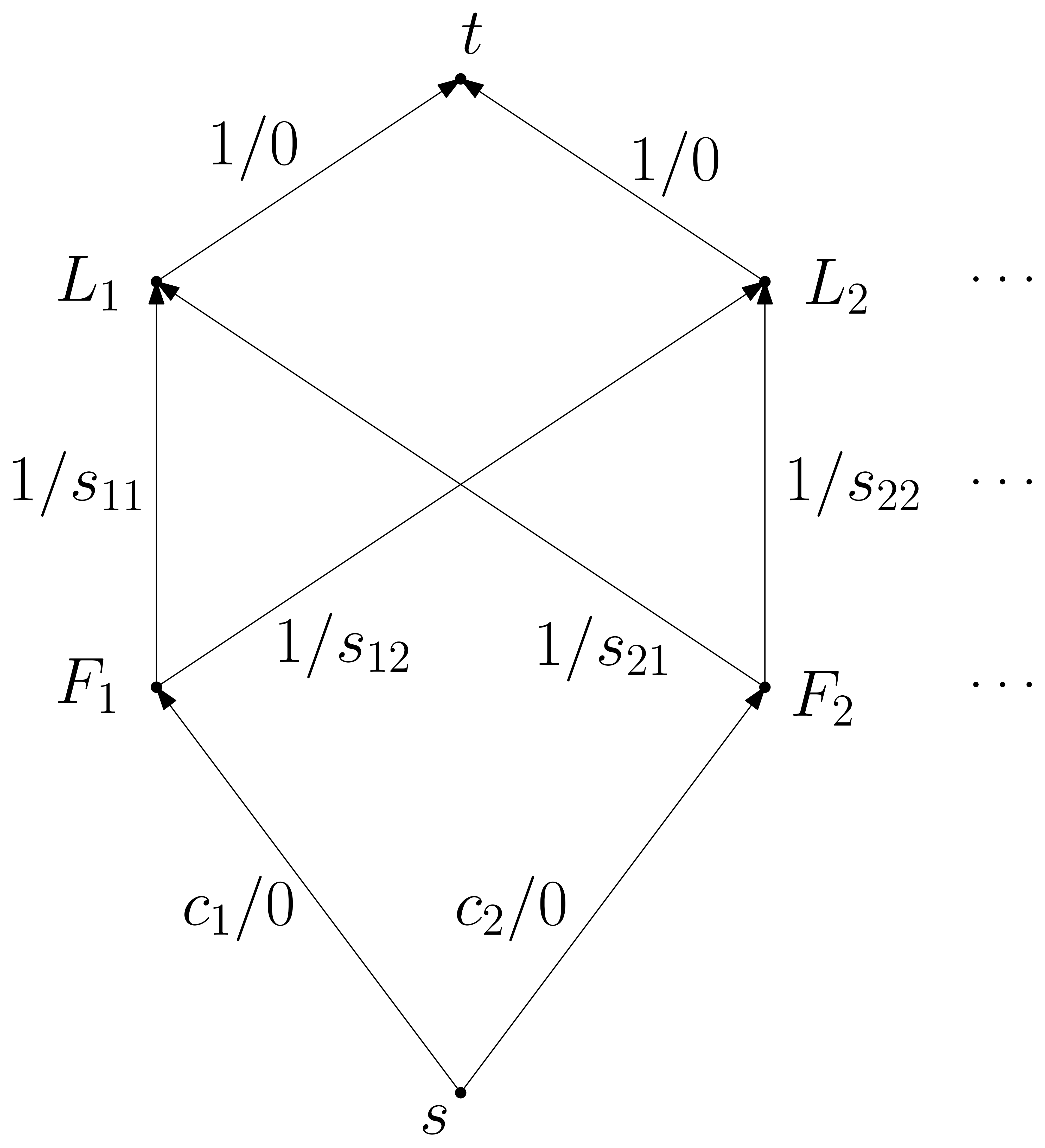
Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_j s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i: \sum_j y_{ij} \leq x_i c_i$$

$$\forall \text{clients } j: \sum_i y_{ij} = 1$$

$$y_{ij} \in \{0,1\}$$



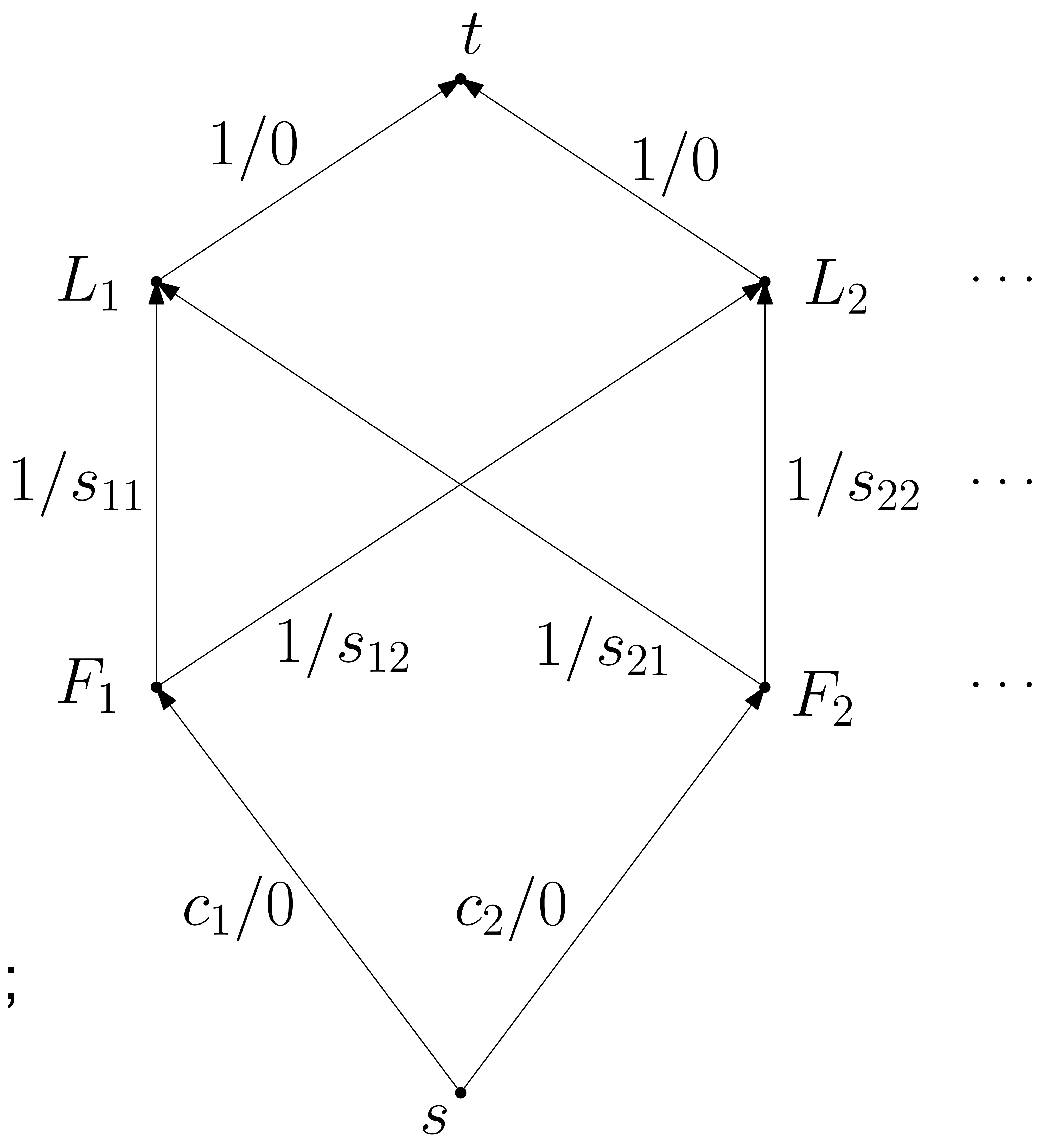
Concrete subproblem:

$$\min \sum_{i:x_i=1} \sum_j s_{ij} y_{ij} \text{ s.t.}$$

$$\forall \text{facilities } i: \sum_j y_{ij} \leq x_i c_i$$

$$\forall \text{clients } j: \sum_i y_{ij} = 1$$

$$y_{ij} \in \{0,1\}$$



With fixed x_i, reduces to Min-Cost Max-s, t-Flow (in P); can be solved as LP and will automatically be integer.

Third Decomposition Approach: Benders' Decomposition

Classical Benders' Decomposition (if the subproblem is an LP):

Third Decomposition Approach: Benders' Decomposition

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem

Classical Benders’ Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem
- Two options: feasible or infeasible

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem
- Two options: feasible or infeasible
- If feasible, we get:

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem
- Two options: feasible or infeasible
- If feasible, we get:
  - a solution (check if it's the new best solution),

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem
- Two options: feasible or infeasible
- If feasible, we get:
  - a solution (check if it's the new best solution),
  - multiplying each constraint with its dual value, adding them up and reordering:

$\theta \geq \alpha - \sum_i \beta_i x_i$ (optimality cut over the master variables x_i)

Classical Benders' Decomposition (if the subproblem is an LP):

- Solve the master
- With the fixed master variables, solve the subproblem
- Two options: feasible or infeasible
- If feasible, we get:
  - a solution (check if it's the new best solution),
  - multiplying each constraint with its dual value, adding them up and reordering:

$\theta \geq \alpha - \sum_i \beta_i x_i$ (optimality cut over the master variables x_i)

- If infeasible, we get a Farkas' lemma certificate (can be turned into a feasibility cut)

Details — Benders' Optimality Cuts

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

write every subproblem constraint as $\sum_j a_{ij} y_j \simeq b_i - \sum_k t_{ik} x_k \qquad \simeq \in \{ \leq , = , \geq \}$

Details — Benders' Optimality Cuts

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

write every subproblem constraint as $\sum_j a_{ij} y_j \simeq b_i - \sum_k t_{ik} x_k \qquad \simeq \in \{ \leq , = , \geq \}$

multiply each constraint with its optimal dual coefficient u_i and sum them up:

Details — Benders' Optimality Cuts

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

write every subproblem constraint as $\sum_j a_{ij} y_j \simeq b_i - \sum_k t_{ik} x_k \qquad \simeq \in \{ \leq, =, \geq \}$

multiply each constraint with its optimal dual coefficient u_i and sum them up:

$$\sum_i u_i \left(\sum_j a_{ij} y_j \right) \geq \sum_i u_i b_i - \sum_i u_i \left(\sum_k t_{ik} x_k \right)$$

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

write every subproblem constraint as $\sum_j a_{ij} y_j \simeq b_i - \sum_k t_{ik} x_k$ $\quad\quad \simeq \in \{ \leq , = , \geq \}$

multiply each constraint with its optimal dual coefficient u_i and sum them up:

$$\sum_i u_i \left(\sum_j a_{ij} y_j \right) \geq \sum_i u_i b_i - \sum_i u_i \left(\sum_k t_{ik} x_k \right)$$

dual feasibility, reordering: $\sum_j c_j y_j \geq \sum_j \left(\sum_i u_i a_{ij} \right) y_j \geq \sum_i u_i b_i - \sum_i u_i \left(\sum_k t_{ik} x_k \right)$

With objective $\min \sum_k d_k x_k + \sum_j c_j y_j$:

write every subproblem constraint as $\sum_j a_{ij} y_j \simeq b_i - \sum_k t_{ik} x_k$ $\qquad \simeq \in \{ \leq , = , \geq \}$

multiply each constraint with its optimal dual coefficient u_i and sum them up:

$$\sum_i u_i \left(\sum_j a_{ij} y_j \right) \geq \sum_i u_i b_i - \sum_i u_i \left(\sum_k t_{ik} x_k \right)$$

dual feasibility, reordering: $\sum_j c_j y_j \geq \sum_j \left(\sum_i u_i a_{ij} \right) y_j \geq \sum_i u_i b_i - \sum_i u_i \left(\sum_k t_{ik} x_k \right)$

Replacing the RHS by θ yields a (potentially violated) cut.
If it is not violated, we are done (proof via complementary slackness).

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Inference dual: what is the best bound on the objective that can be inferred (by some proof system, which may include search) from the constraints?

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Inference dual: what is the best bound on the objective that can be inferred (by some proof system, which may include search) from the constraints?

Much stronger than weighted linear combination of linear constraints!

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Inference dual: what is the best bound on the objective that can be inferred (by some proof system, which may include search) from the constraints?

Much stronger than weighted linear combination of linear constraints!

But:

Details — Generalization: Logic-based Benders' Decomposition

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Inference dual: what is the best bound on the objective that can be inferred (by some proof system, which may include search) from the constraints?

Much stronger than weighted linear combination of linear constraints!

But:

- There is no universal cut recipe (except for trivial no-goods that do not scale in practice)

Our cut recipe needs the subproblem to be an LP.

LBBD relaxes that requirement: could be a CP, SAT, MIP, DP, …

The cut is instead derived 'logically' from the *inference dual*

Inference dual: what is the best bound on the objective that can be inferred (by some proof system, which may include search) from the constraints?

Much stronger than weighted linear combination of linear constraints!

But:

- There is no universal cut recipe (except for trivial no-goods that do not scale in practice)
- Problem-specific certificates from the subproblem needed (e.g., infeasible core, dual, …)

Conclusion

Seen decomposition ideas:

- Lagrangian relaxation (dualize away coupling constraints)
- Dantzig-Wolfe & Column Generation idea (solutions to local subproblems → variables)
- Benders' decomposition (approx. objective in master refined by cuts, LP subproblem)
- Logic-based Benders' decomposition (more general subproblem, inferred cuts)

General note: such approaches often require a lot of tuning to reach their full potential.

So, before a serious implementation attempt, look up the details to keep in mind!