Using Functional Programming to recognize Named Structure in an Optimization Problem: Application to Pooling

Francesco Ceccon
francesco.ceccon14@imperial.ac.uk

Ruth Misener
r.misener@imperial.ac.uk

Department of Computing
Imperial College London

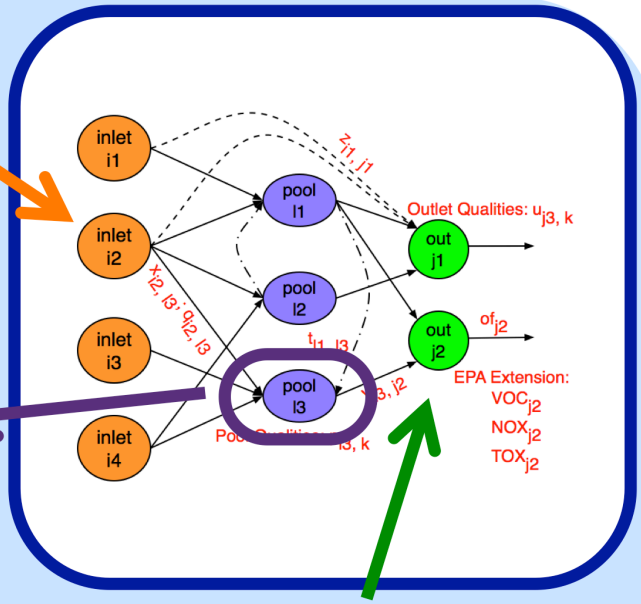
Aug 2016

The Pooling Problem

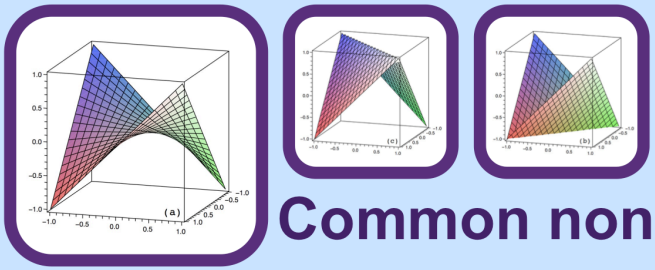
Know input species concentrations (*e.g.*, sulfur)

Variables

$$\sum_{i \in \mathrm{In}} c_{\mathrm{Sulfur},\, i} \cdot f_i = c_{\mathrm{Sulfur,\, Out}} \cdot \sum_{o \in \mathrm{Out}} f_o$$



Have output species requirements (e.g., laws)



Common nonlinear term: x × y

Ceccon, Kouyialis, Misener. Using Functional Programming to recognize Named Structure in an Optimization Problem: Application to Pooling. *AIChE Journal*; DOI 10.1002/aic.15308, 2016.

Why is the local structure of a pooling problem important?

- Can construct convex relaxations and feasibility-based bounds tightening in multiple ways (Tawarmalani, Ahmed, and Sahinidis 2002)

$$\mathbf{p}_{\ell,\mathbf{k}} \cdot \sum_{j \in J} \mathbf{f}_{\ell,\mathbf{j}} = \sum_{j \in J} \mathbf{p}_{\ell,\mathbf{k}} \cdot \mathbf{f}_{\ell,\mathbf{j}}$$

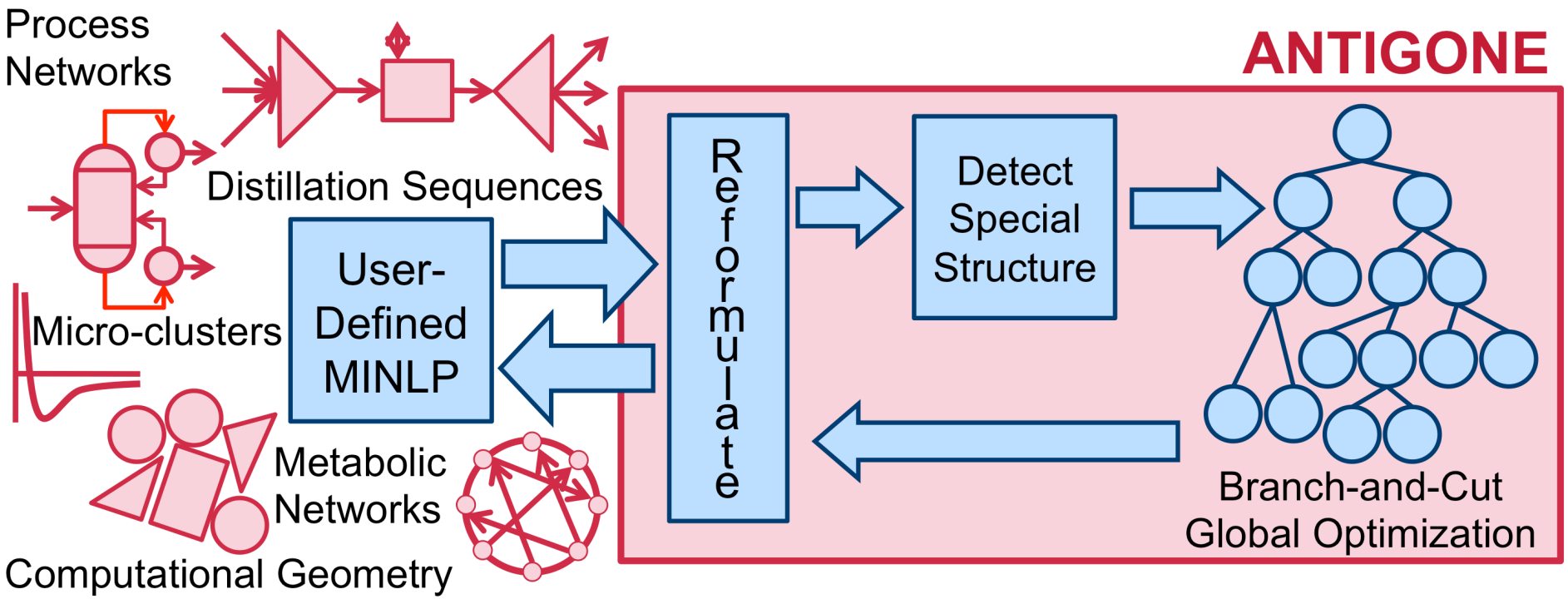
- More likely to find the Reduction Constraint linear equality equation/variable products (Tawarmalani and Sahinidis 2002; Liberti and Pantelides 2006)

$$\left(\sum_j a_{m,j} \cdot x_j - b_m \right) \cdot \mathbf{x_i} = \sum_j a_{m,j} \cdot x_j \cdot \mathbf{x_i} - b_m \cdot \mathbf{x_i} = 0$$

- Loops in the undirected graph representation of an expression collecting the nonlinear terms (Misener, Smadbeck, and Floudas 2015)

$$\sum_{i \in I, j \in J} x_i x_j$$

Globally Optimizing MINLP: Computational System



ANTIGONE

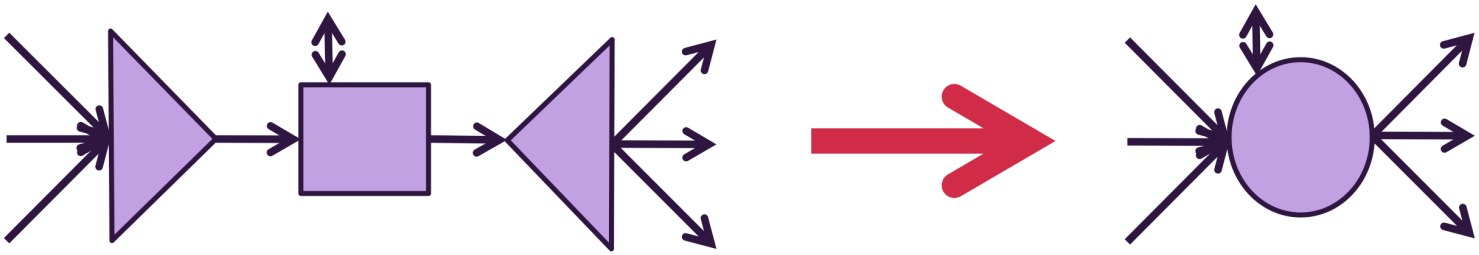
Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations [Misener, Floudas, GAMS GmbH]

Misener & Floudas, ANTIGONE, *J Global Optimization*; 2014

Energy Systems: Reformulating User Input

$> 10\times$ Speed-up for 10% of process networks

ANTIGONE, BARON, Couenne, LINDO



Mixer

$$\begin{cases} \sum_i F_{M,i} = F_T^{IN} \\ \sum_i x_{M,i,j} \cdot F_{M,i} = x_{T,j}^{IN} \cdot F_T^{IN} \quad \forall\, j \in \{1, \ldots, J\} \end{cases}$$

Treatment

$$\begin{cases} F_T^{IN} = F_T^{OUT} \\ x_{T,j}^{IN} = \beta_{T,j} \cdot x_{T,j}^{OUT} \quad \forall\, j \in \{1, \ldots, J\} \end{cases}$$

Splitter

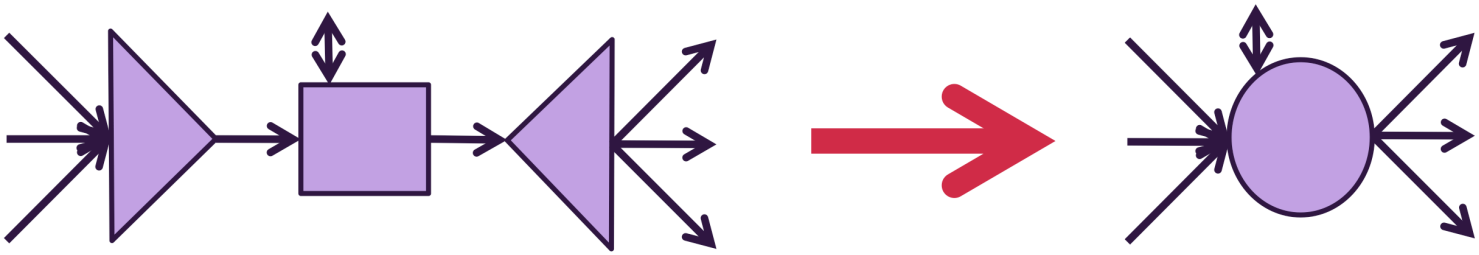
$$\begin{cases} F_T^{OUT} = \sum_i F_{S,i} \\ x_{T,j}^{OUT} = x_{T,i,j} \quad \forall\, i \in \{1, \ldots, I\};\ j \in \{1, \ldots, J\} \end{cases}$$

Misener, Smadbeck & Floudas, *Optim Met Softw*, 2015

Energy Systems: Reformulating User Input

$> 10\times$ Speed-up for 10% of process networks

ANTIGONE, BARON, Couenne, LINDO



Mixer

$$\begin{cases} \sum_i F_{M,i} = F_T^{IN} \\ \sum_i x_{M,i,j} \cdot F_{M,i} = x_{T,j}^{IN} \cdot F_T^{IN} \quad \forall\, j \in \{1, \ldots, J\} \end{cases}$$

Treatment

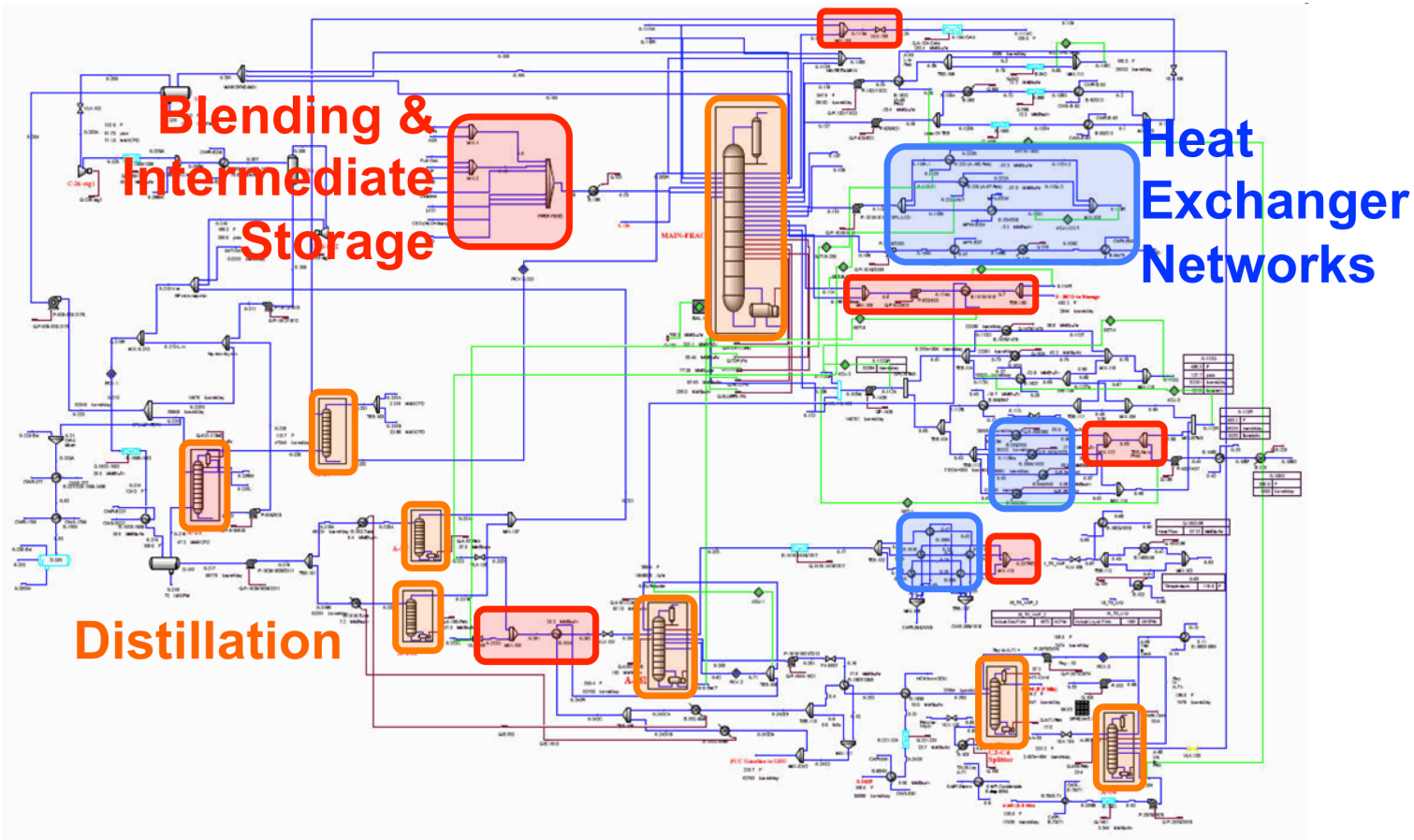
$$\begin{cases} F_T^{IN} = F_T^{OUT} \\ x_{T,j}^{IN} = \beta_{T,j} \cdot x_{T,j}^{OUT} \quad \forall\, j \in \{1, \ldots, J\} \end{cases}$$

Splitter

$$\begin{cases} F_T^{OUT} = \sum_i F_{S,i} \\ x_{T,j}^{OUT} = x_{T,i,j} \quad \forall\, i \in \{1, \ldots, I\};\ j \in \{1, \ldots, J\} \end{cases}$$

Misener, Smadbeck & Floudas, *Optim Met Softw*, 2015

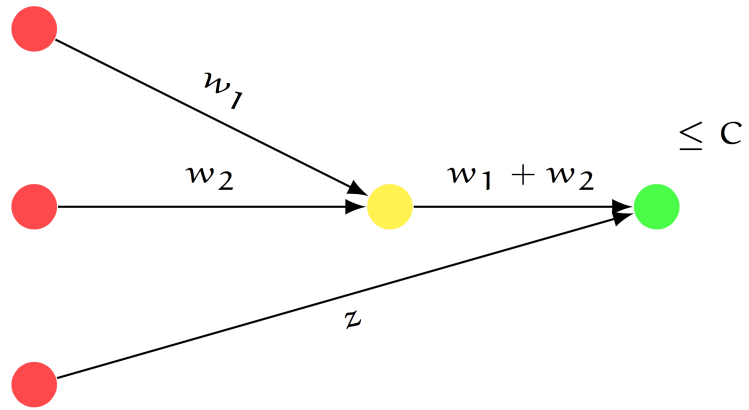
Need to find pooling structure . . .



- Pool-to-pool loops; network structure is not feed-forward;
- Nonlinear blending rules for octane number, etc.;
- Disjunctive decisions.

More network structure . . .

Extract valid inequalities for small pooling problems (D'Ambrosio et al. 2014)



Uncover sparsity and piecewise structure (Baltean-Lugojan and Misener 2016)

Discretize to get a nice MILP approximation (Dey and Gupte 2015)

Exploring the P/NP boundary (Alfaki and Haugland 2013; Boland, Kalinowski, and Rigterink 2015; Haugland 2015; Baltean-Lugojan and Misener 2016)

Prior Work: Finding MILP Patterns

(Brown and Wright 1984; Roy and Wolsey 1987; Bixby and Fourer 1988; Nemhauser, Savelsbergh, and Sigismondi 1994; Gulpinar et al. 2004; Achterberg and Raack 2010; Salvagnin 2016)

Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition
$$v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J$$

Simplex
$$\sum_{i\in I} q_{il} = 1 \quad \forall l \in L$$

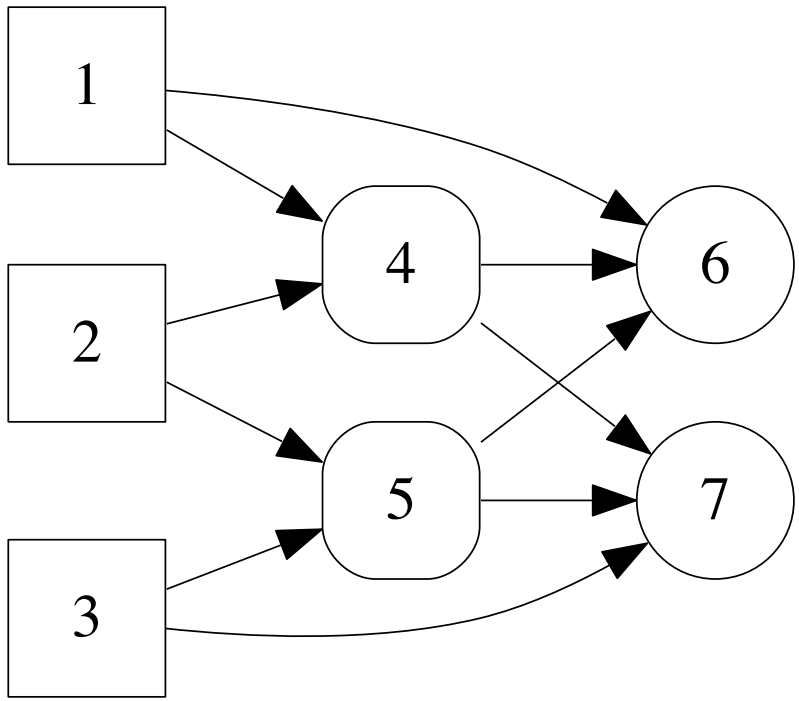
Reduction
$$\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J$$

Input Capacity
$$\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I$$

Pool Capacity
$$\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L$$

Output Capacity
$$\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J$$

Product Quality
$$\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J$$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j)\mathbf{y_{ij}} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j)\mathbf{v_{ilj}}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

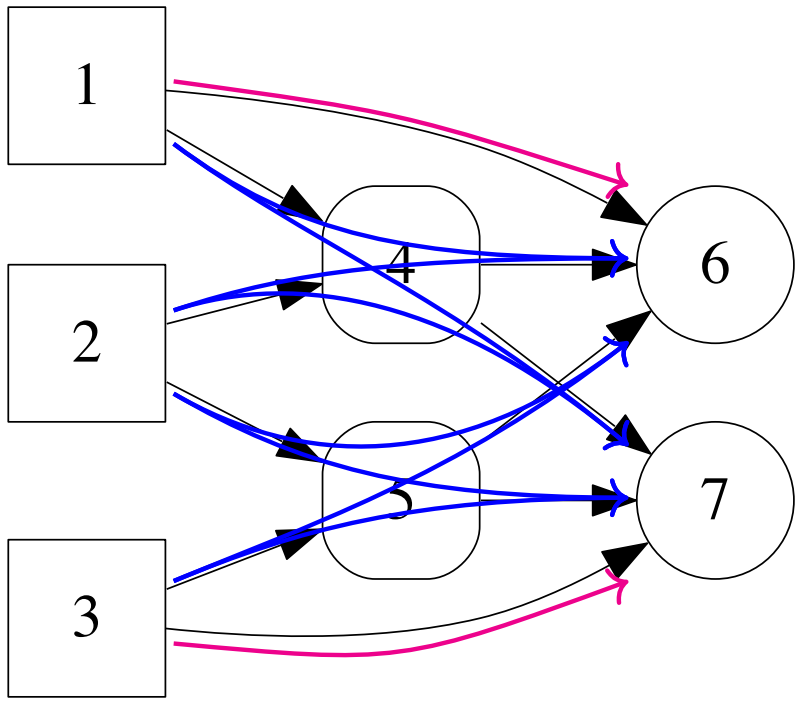
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

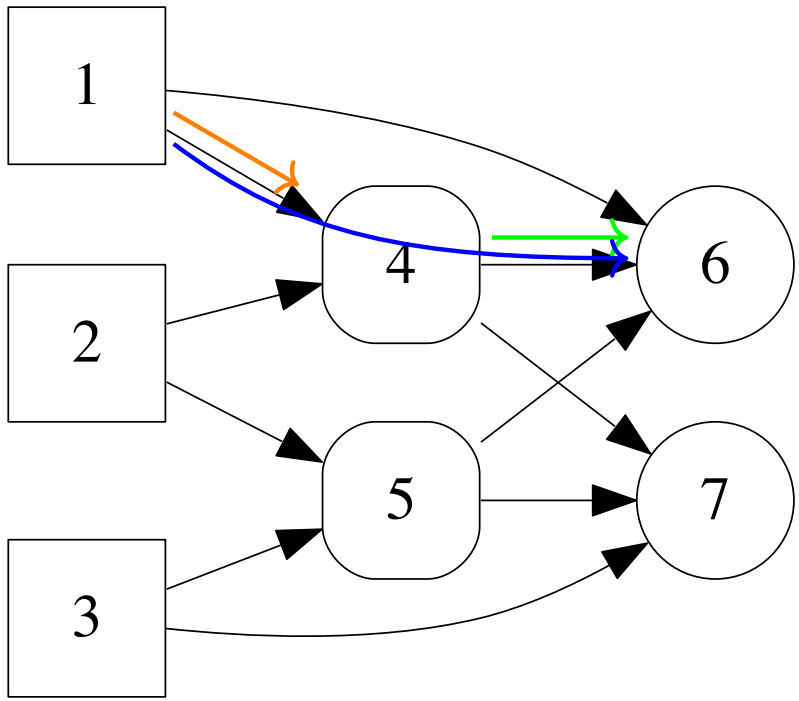
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} \mathbf{q_{il}} = 1 \quad \forall l \in L \right.$

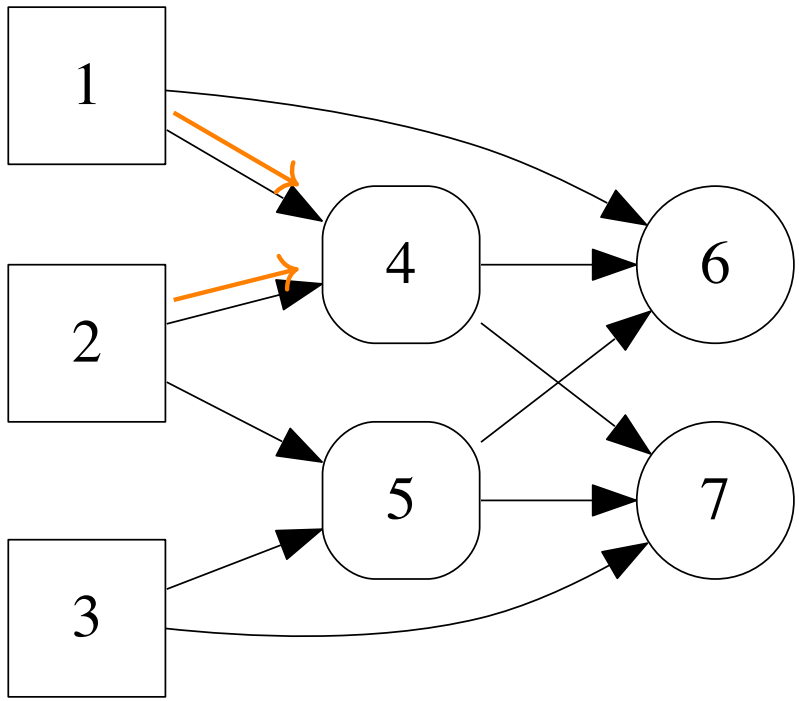
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

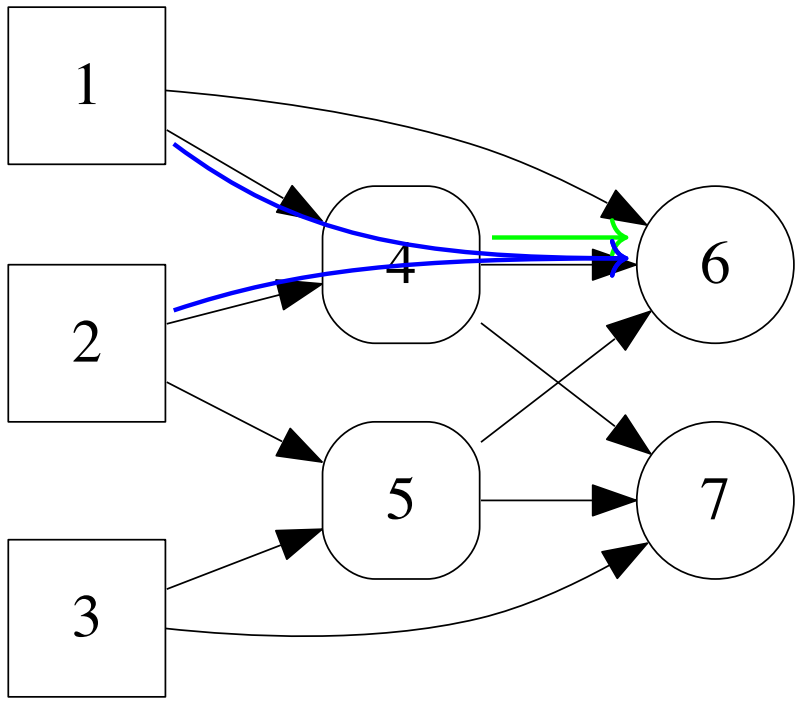
Reduction $\left[\sum_{i\in I} \mathbf{v_{ilj}} = \mathbf{y_{lj}} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

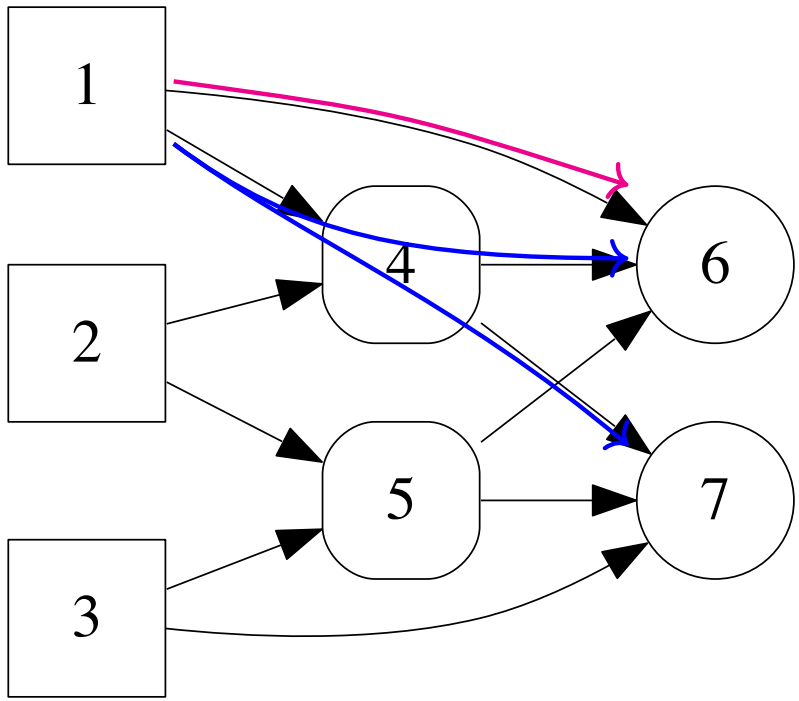
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \leq c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \leq c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \leq c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \leq P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \geq P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

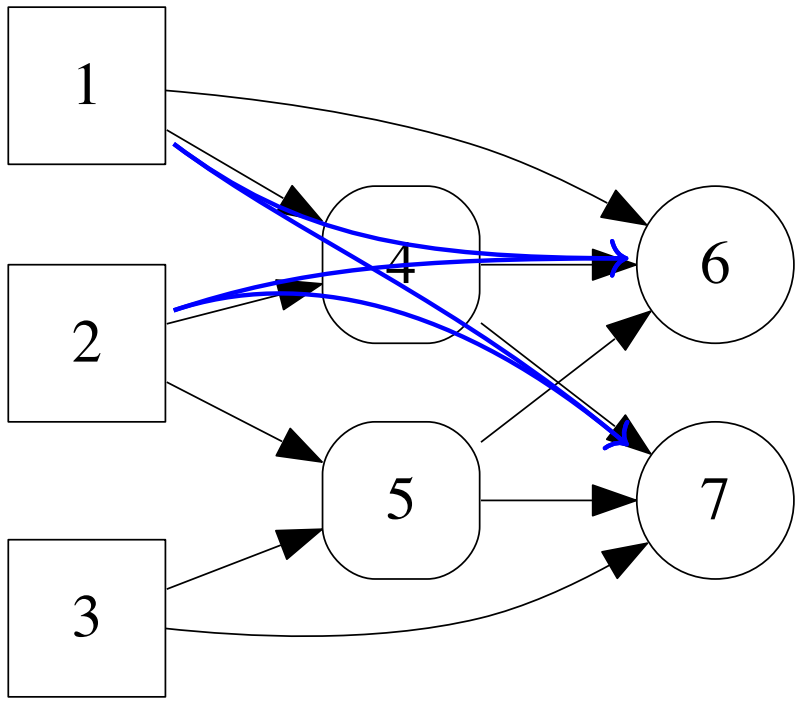
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} \mathbf{v_{ilj}} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

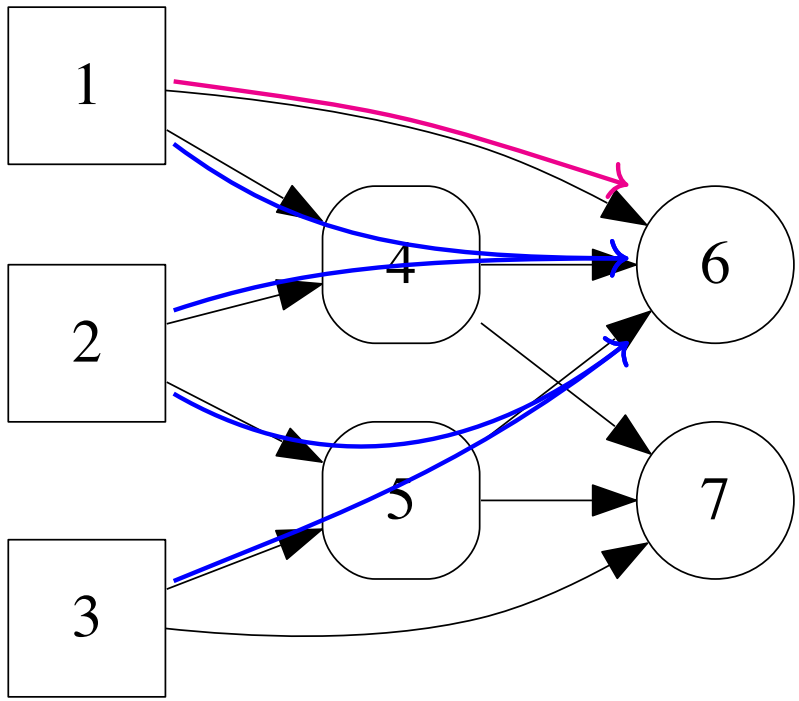
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} \mathbf{y_{ij}} + \sum_{i\in I, l\in L} \mathbf{v_{ilj}} \le c_j \quad \forall j \in J \right.$

Product Quality $\left[\sum_{i\in I} C_{ik} y_{ij} + \sum_{i\in I, l\in L} C_{ik} v_{ilj} \begin{cases} \le P^U_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \right) \end{cases} \quad \forall k \in K, j \in J \right.$



Canonical Formulation

Objective $\min_{y,v,q} \sum_{i\in I, j\in J} (c_i - d_j) y_{ij} + \sum_{i\in I, l\in L, j\in J} (c_i - d_j) v_{ilj}$

Path Definition $\left[v_{ilj} = q_{il} y_{lj} \quad \forall i \in I, l \in L, j \in J \right.$

Simplex $\left[\sum_{i\in I} q_{il} = 1 \quad \forall l \in L \right.$

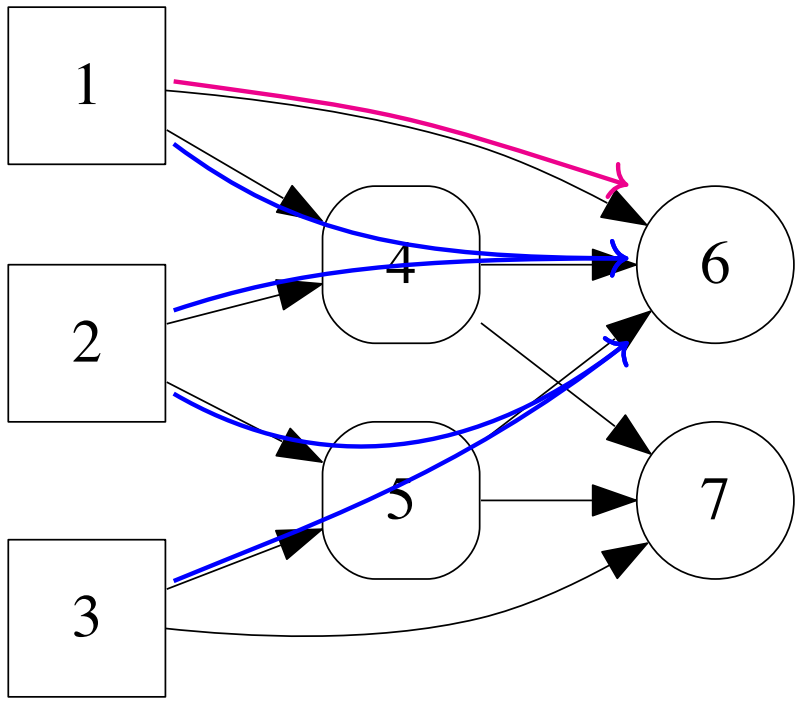
Reduction $\left[\sum_{i\in I} v_{ilj} = y_{lj} \quad \forall l \in L, j \in J \right.$

Input Capacity $\left[\sum_{j\in J} y_{ij} + \sum_{l\in L, j\in J} v_{ilj} \le c_i \quad \forall i \in I \right.$

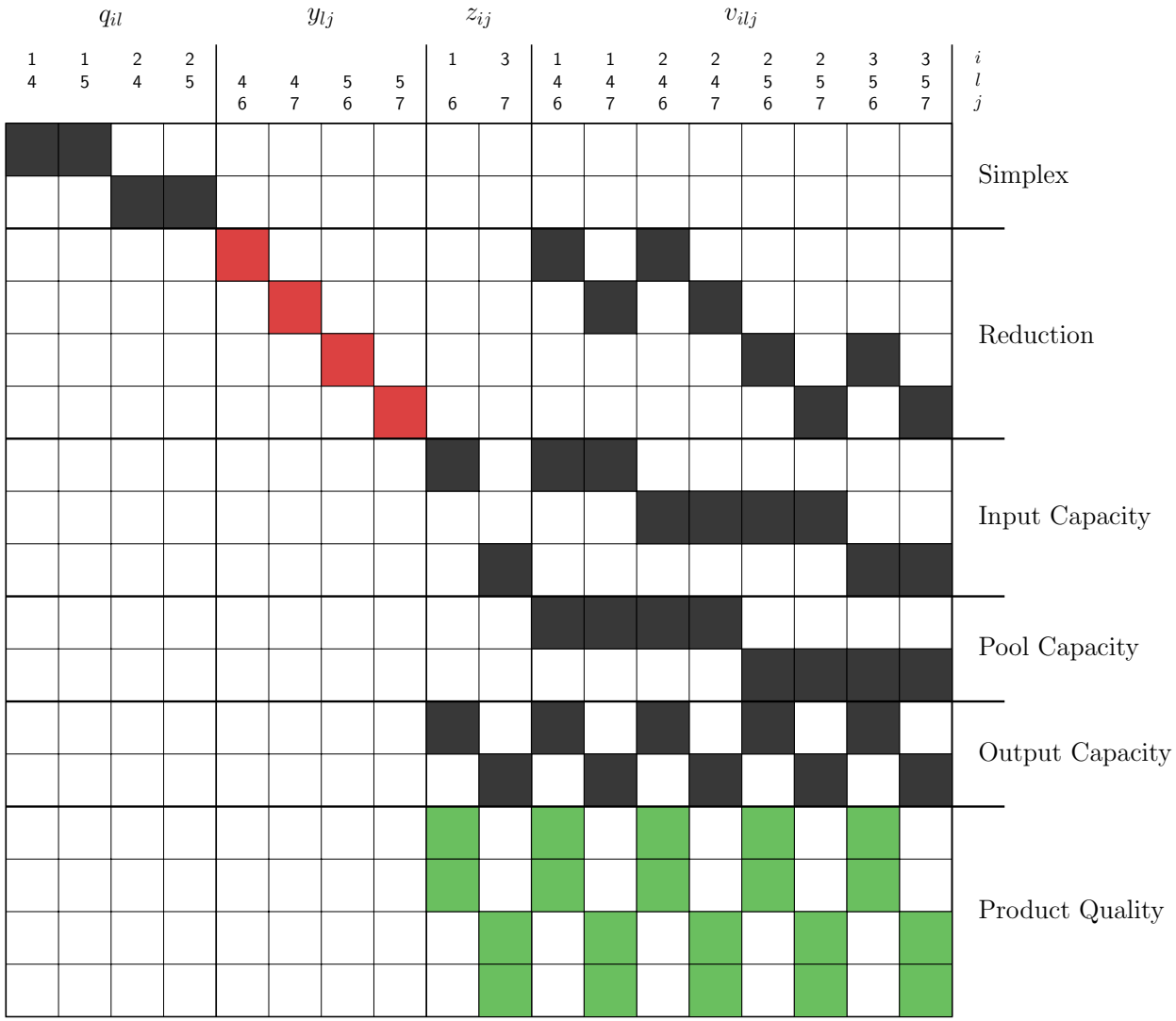
Pool Capacity $\left[\sum_{i\in I, j\in J} v_{ilj} \le c_l \quad \forall l \in L \right.$

Output Capacity $\left[\sum_{i\in I} y_{ij} + \sum_{i\in I, l\in L} v_{ilj} \le c_j \quad \forall j \in J \right.$

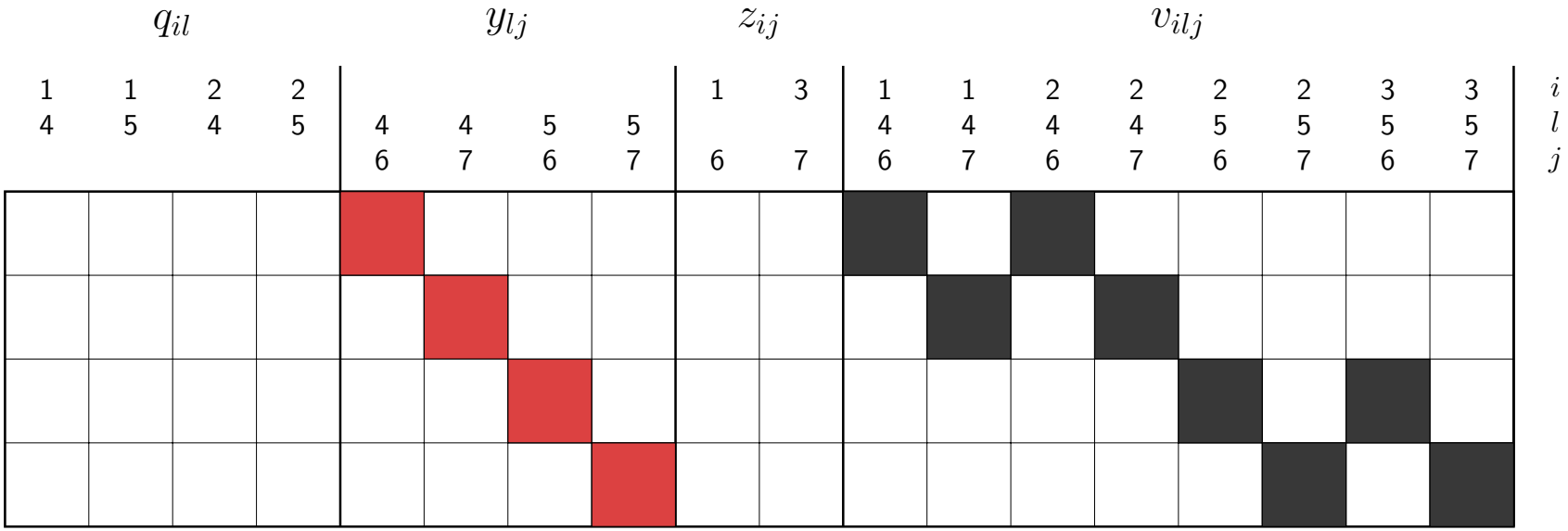
Product Quality $\left[\sum_{i\in I} C_{ik} \mathbf{y_{ij}} + \sum_{i\in I, l\in L} C_{ik} \mathbf{v_{ilj}} \left\{ \begin{array}{l} \le P^U_{jk} \left(\sum_{i\in I} \mathbf{y_{ij}} + \sum_{i\in I, l\in L} \mathbf{v_{ilj}} \right) \\ \ge P^L_{jk} \left(\sum_{i\in I} \mathbf{y_{ij}} + \sum_{i\in I, l\in L} \mathbf{v_{ilj}} \right) \end{array} \right. \right. \quad \forall k \in K, j \in J$



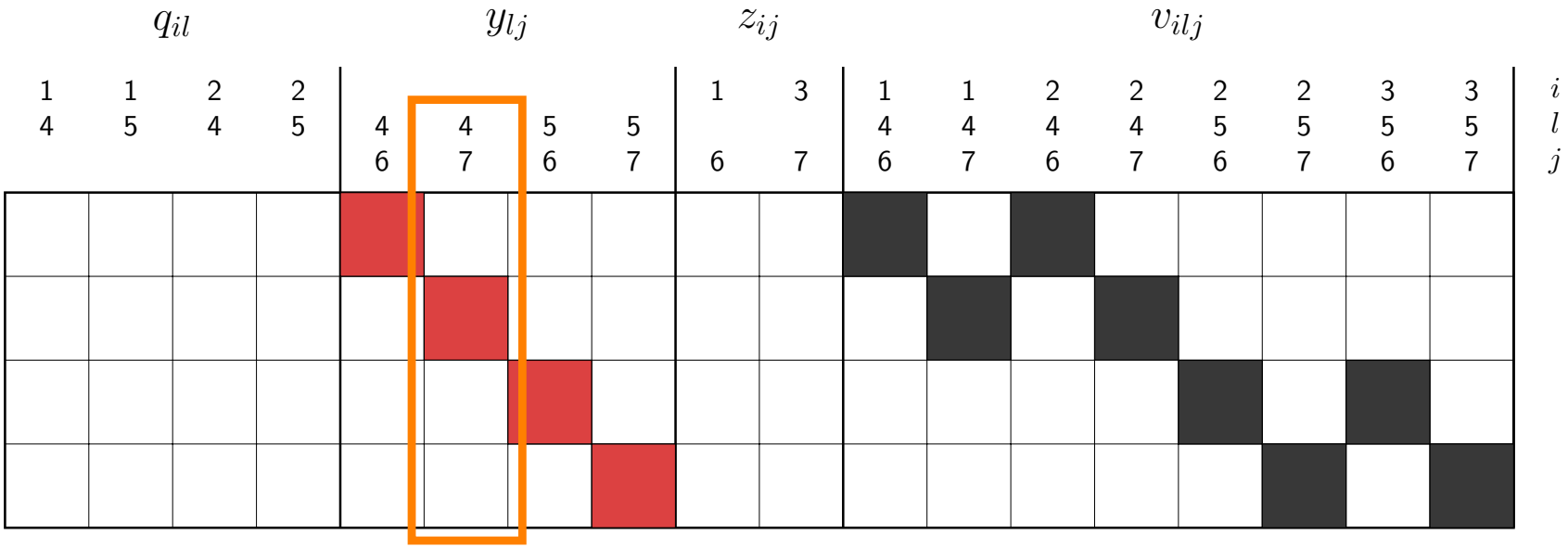
Visualizing the Pooling Problem



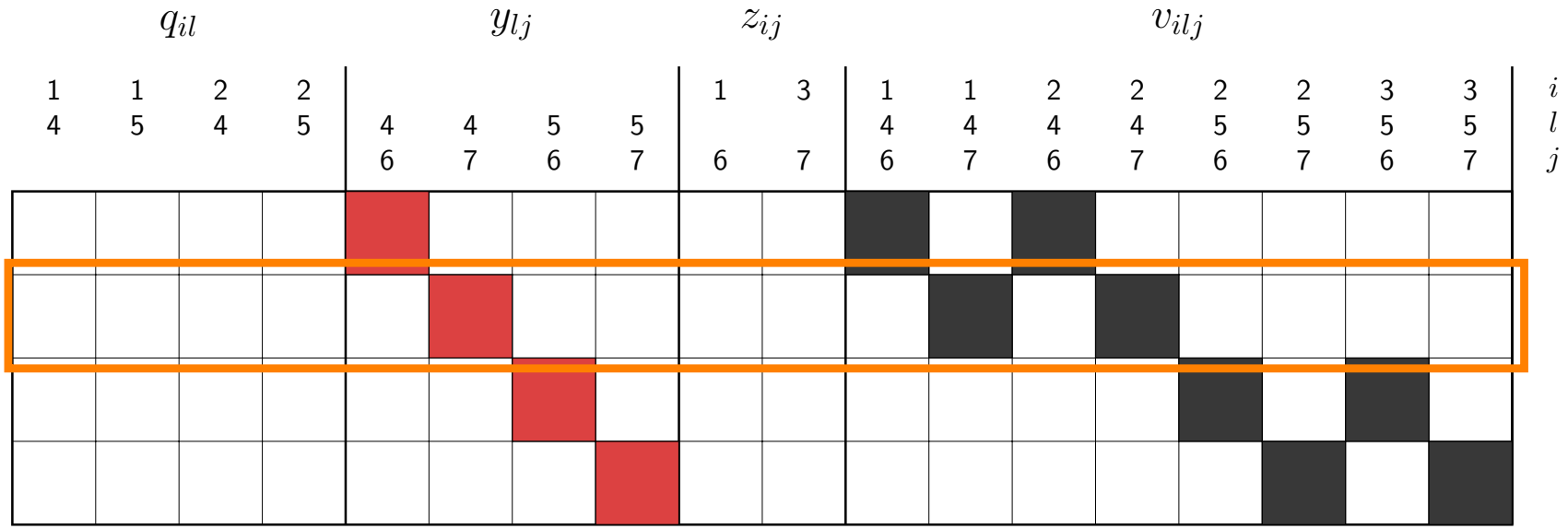
Visualizing the Pooling Problem



Visualizing the Pooling Problem



Visualizing the Pooling Problem



$$v_{147} + v_{247} - y_{47} = 0$$

Overview

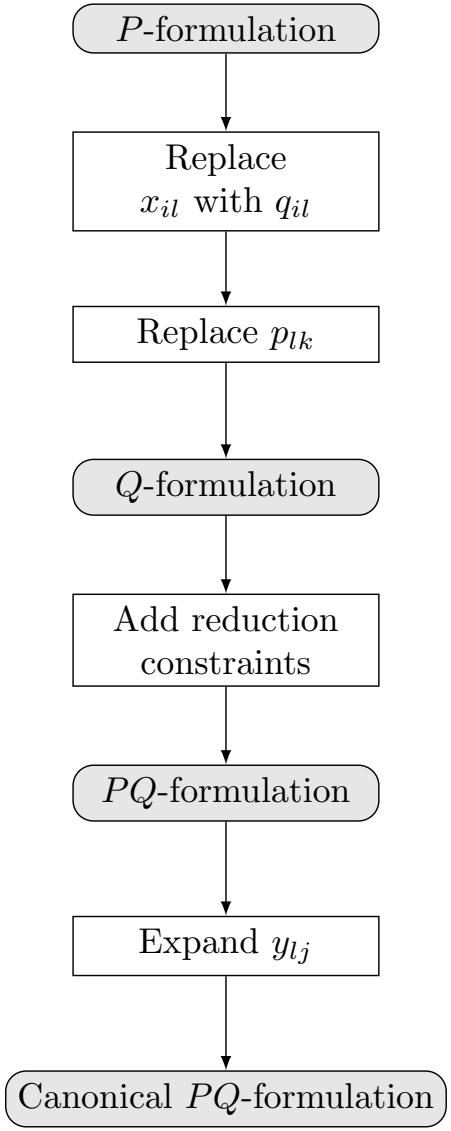
The proposed method can be divided in 3 phases:

- Transforming to canonical PQ-formulation
- Identifying constraints
- Building the network

Transform

Starting from the P-formulation, we want to obtain the canonical PQ-formulation:

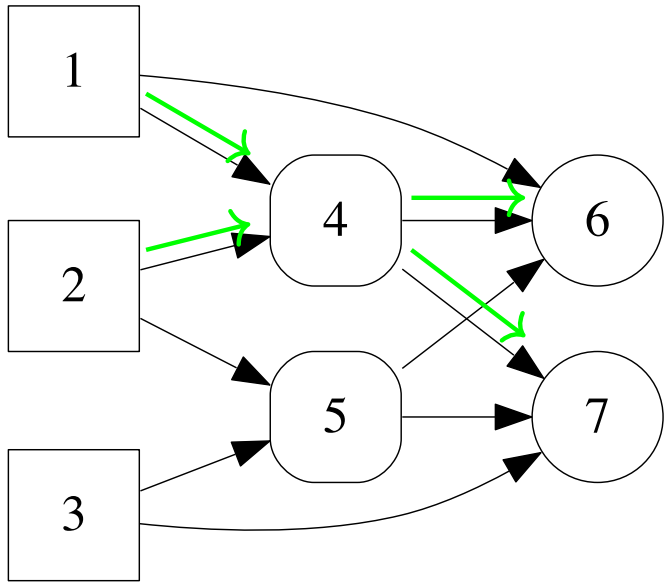
- Replace flow rate y_{il} with proportional flow rate q_{il}
- Add the reduction constraints
- Replace all occurrences of y_{lj} outside of reductions



Replacing Flow Rates

Find material balance constraints

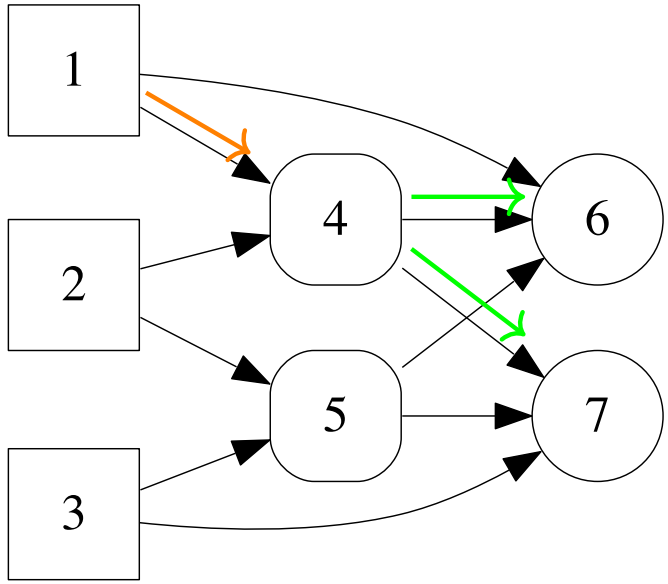
$$\sum_{i \in I} y_{iI} - \sum_{j \in J} y_{Ij} = 0 \quad \forall I \in L$$



Replacing Flow Rates

Replace flow variables y_{ij} with proportional flow variables q_{il}

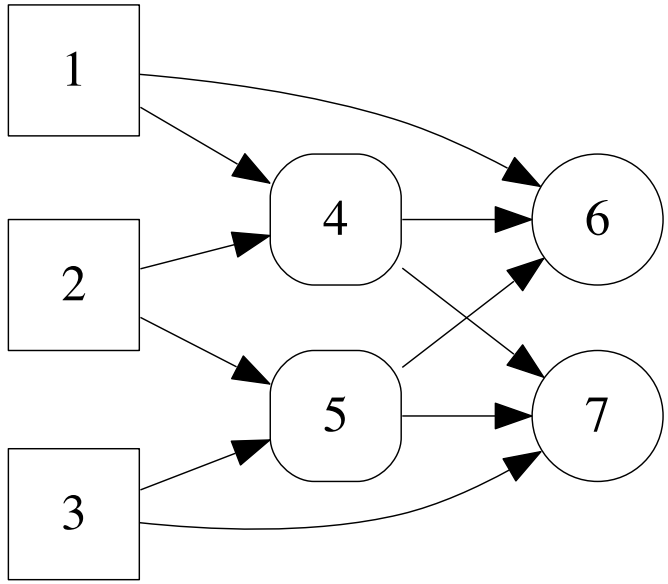
$$y_{il} = q_{il} \sum_{j \in J} y_{lj} \quad \forall i \in I, j \in J$$



Replacing Flow Rates

Finally eliminate variables p_{lk}, obtaining the Q-formulation

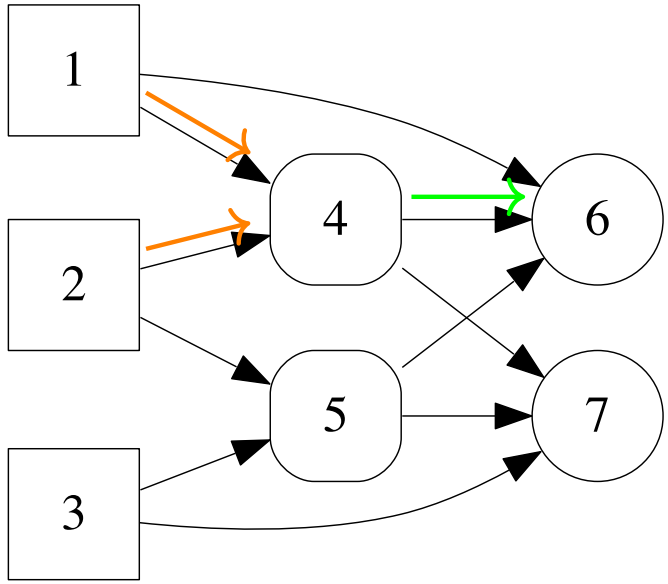
$$p_{lk} = \sum_{i \in I} C_{ik} q_{il}$$



Adding Reductions

For each bilinear term $q_{il}y_{lj}$, add the reduction constraint

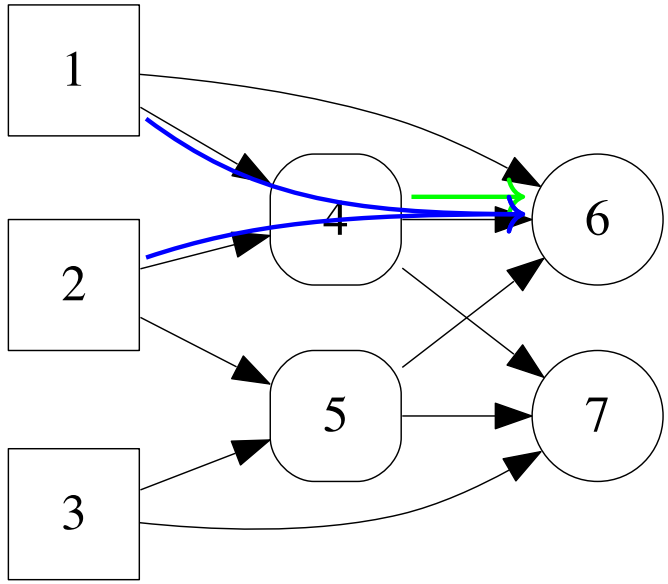
$$\sum_{i \in I} q_{il}y_{lj} = y_{lj} \; \forall l \in L, j \in J$$



Expand y_{lj} Variables

Replace each occurrence of y_{lj} with the equivalent sum of v_{ilj}

$$y_{lj} = \sum_{i \in I} v_{ilj}$$



Identifying Constraints

By looking at patterns that are unique to a certain type of constraint, we can gradually identify all constraints that make a Pooling Problem. We identify the constraints in the following order:

- Reduction & Path Definition Constraints
- Pool Capacities
- Input & Output Capacities
- Quality Constraints

Path Definitions

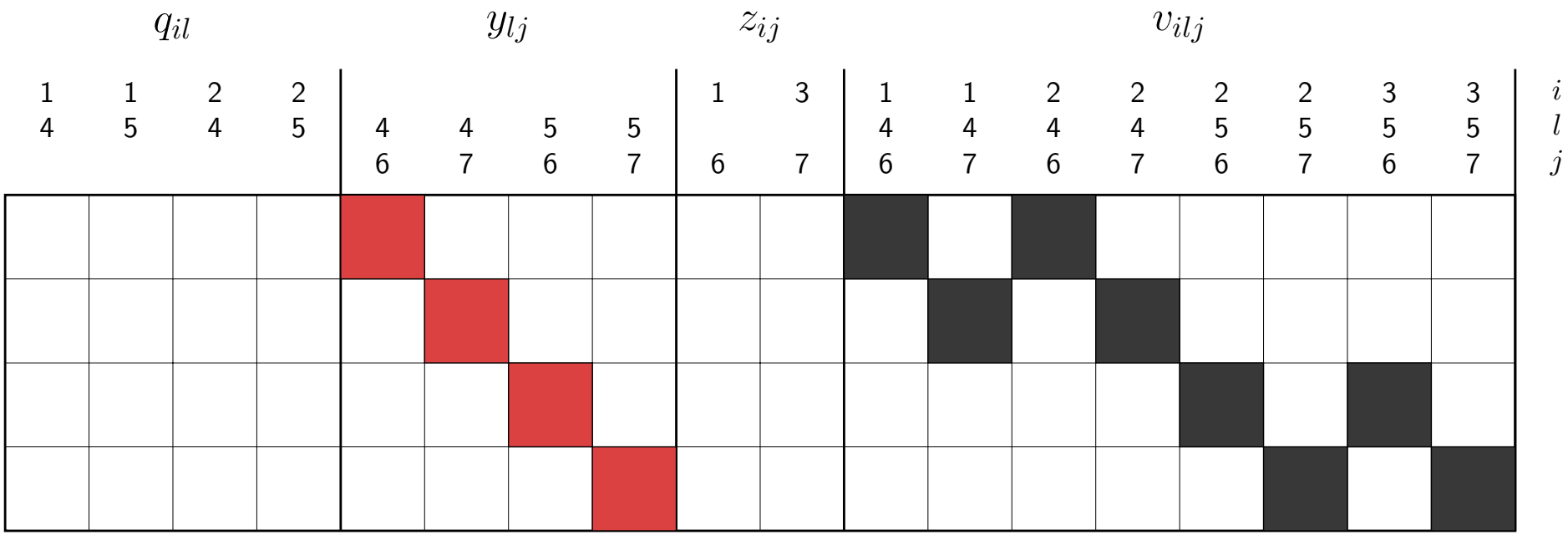
Identify Path Definitions by looking at constraints with one bilinear term $q_{il}y_{lj}$ with coefficient 1, and one linear term v_{ilj} with coefficient -1

$$v_{ilj} = q_{il}y_{lj}$$

Reduction Constraints

If we group Path Definitions by variables y_{lj}, we obtain groups of v_{ilj} with the same (l, j) index. Then we can identify Reduction Constraints

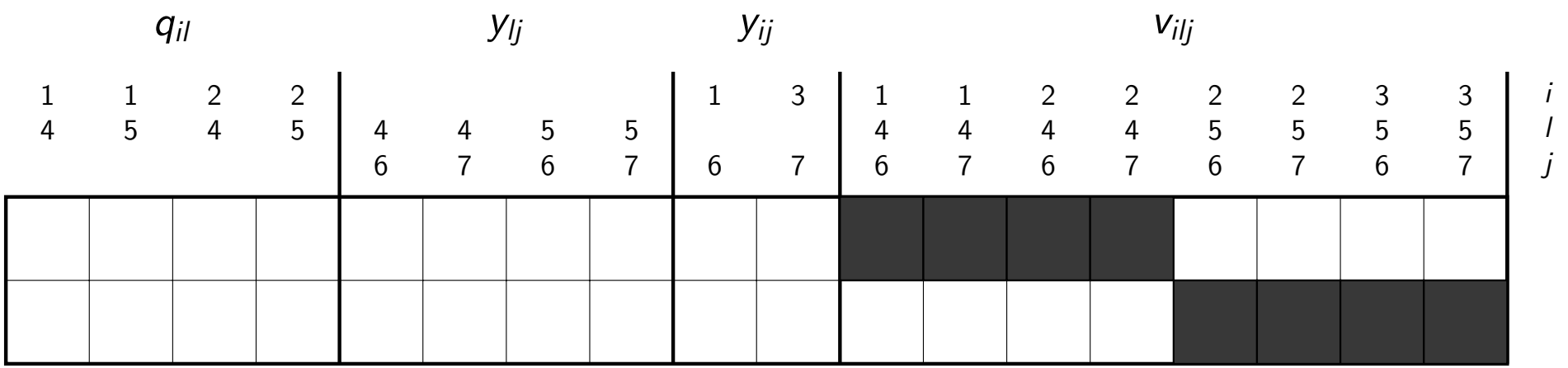
$$\sum_{i \in I} v_{ilj} - y_{lj} = 0$$



Pool Capacities

From path definitions, we notice that the pair q_{il}, y_{lj} must have the same index l. From this, we aggregate all flow variables relative to pool l.

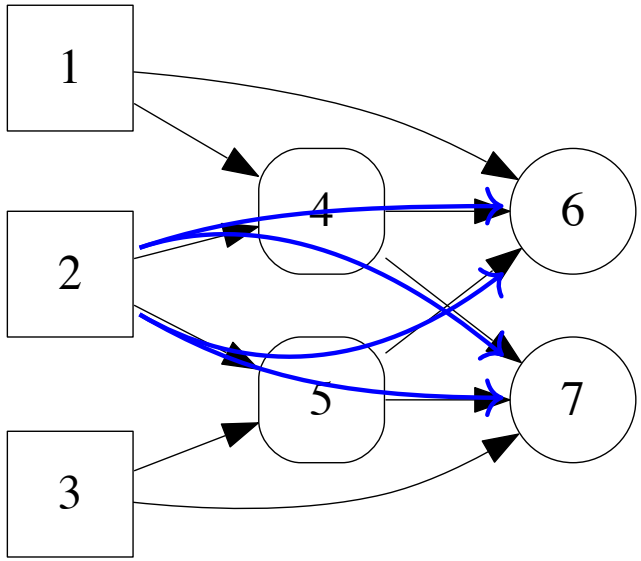
$$\sum_{i \in I, j \in J} v_{ilj} = c_l$$

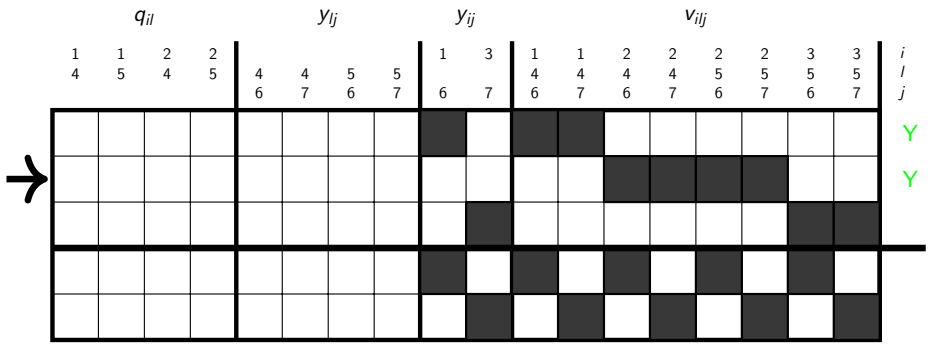


Input Capacities

To find input constraints:

- Group flow variables v_{ilj} by index (l, j)
- Take constraints with maximum one flow variable per group

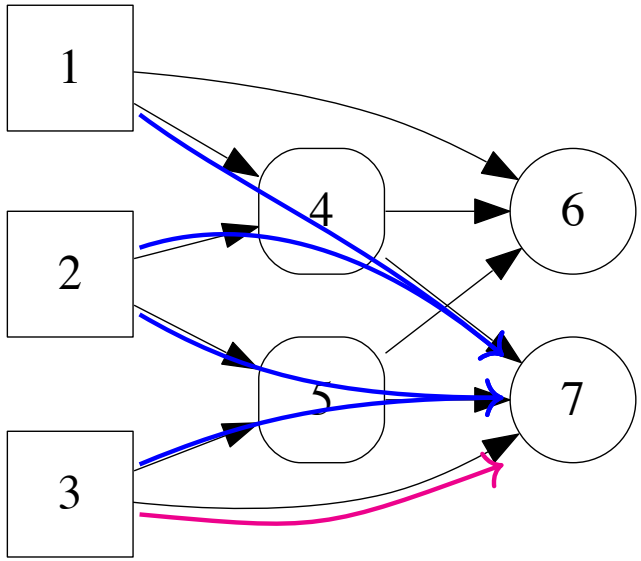


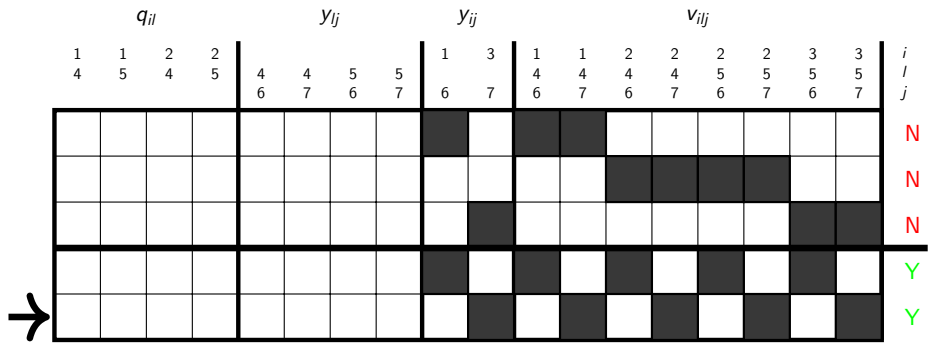


Output Capacities

To find output constraints:

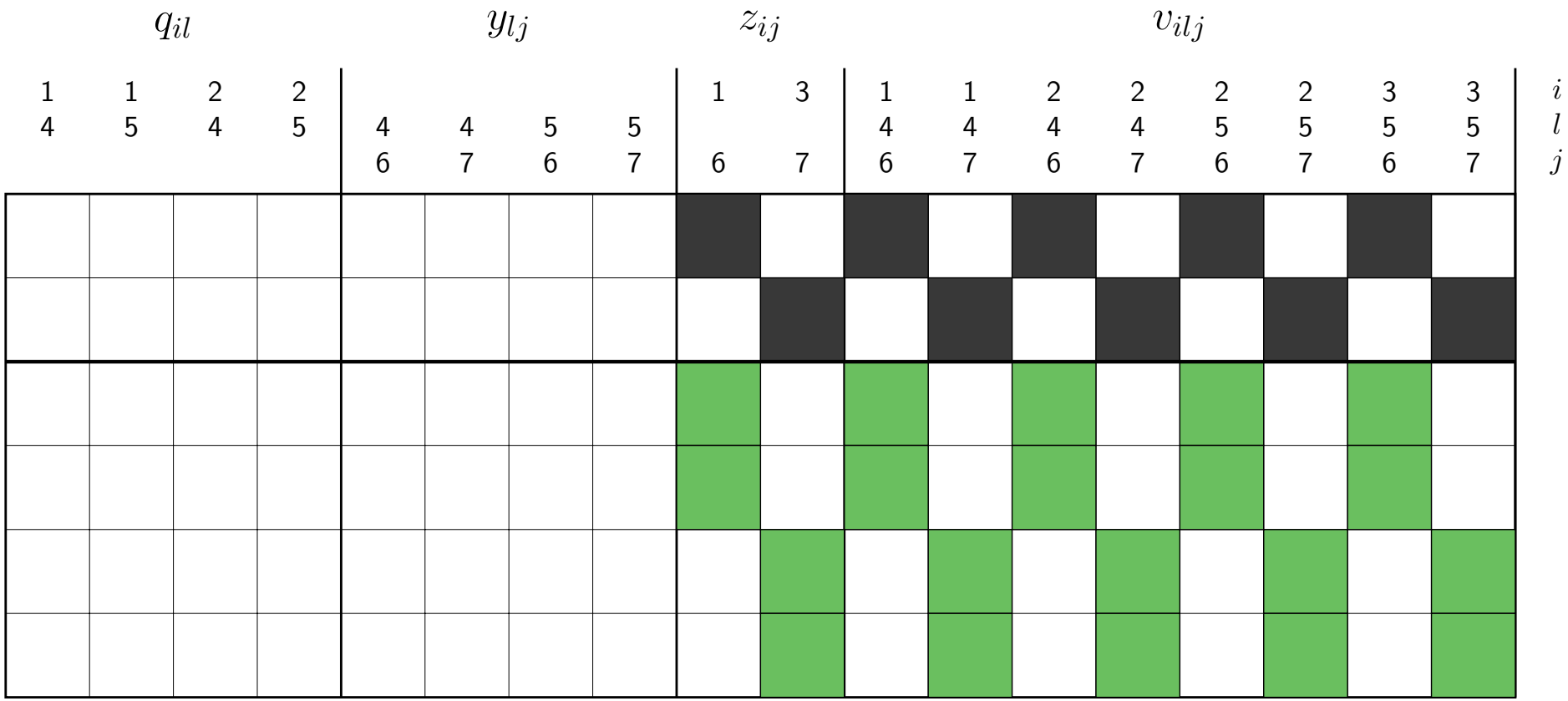
- Group flow variables v_{ilj} by index (i, l)
- Take constraints with maximum one flow variable per group





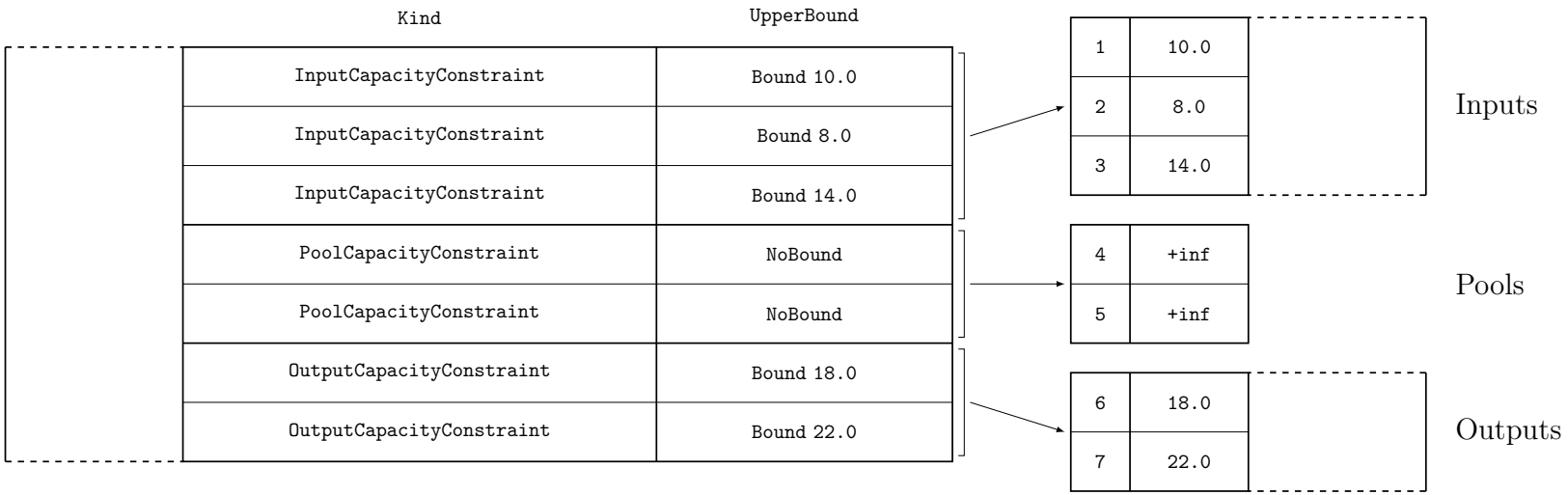
Quality Constraints

Quality Constraints have the same nonzero coefficients as Output Capacities, but with different coefficients values.



Network Nodes & Arcs

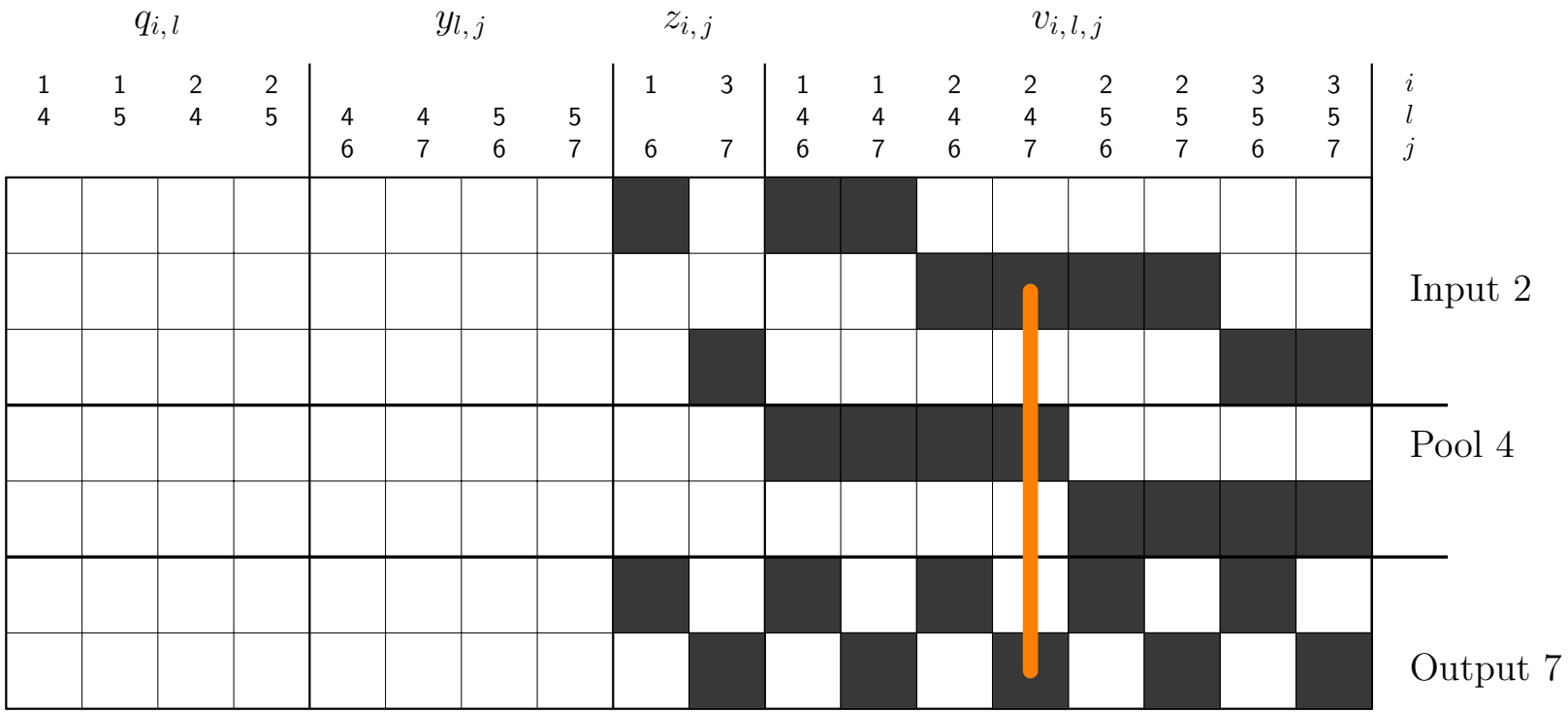
Each Capacity Constraint represents a different node of the network. The flow variables v_{ilj} represent the arcs in the network.



Building nodes from constraints

Network Nodes & Arcs

Each Capacity Constraint represents a different node of the network. The flow variables v_{ilj} represent the arcs in the network.



Adding arcs

Input Cost & Output Revenue

Cost and Revenue constants are found from the problem objective function:

$$\sum_{i \in I, l \in L, j \in J} (d_j - c_i) v_{ilj} - \sum_{i \in I, j \in J} (d_j - c_i) y_{ij} \tag{1}$$

The Linear Program (2) decomposes the values into Input Cost c_i and Output Revenue d_j.

$$\begin{aligned} \min_{c,\, d} \quad & \sum_{i \in I} c_i + \sum_{j \in J} d_j & & \\ \text{s.t.} \quad & c_i - d_j & = \gamma_{i,j} \quad & \forall i \in I, j \in J \\ & c_i & \geq 0 \quad & \forall i \in I \\ & d_j & \geq 0 \quad & \forall j \in J \end{aligned} \tag{2}$$

Input & Output Qualities

Just as parameters c_i, d_j, parameters $C_{ik}, P^L_{jk}, P^U_{jk}$ are not directly present in the canonical PQ-formulation, instead their differences $\Delta^L_{ijk} = C_{ik} - P^L_{jk}$ and $\Delta^U_{ijk} = C_{ik} - P^U_{jk}$ are present:

$$\sum_{i \in I, l \in L} \Delta^L_{ijk} v_{ilj} + \sum_{i \in I} \Delta^L_{ijk} y_{ij} \geq 0 \; \forall j \in J, k \in K$$

$$\sum_{i \in I, l \in L} \Delta^U_{ijk} v_{ilj} + \sum_{i \in I} \Delta^U_{ijk} y_{ij} \geq 0 \; \forall j \in J, k \in K$$

Input & Output Qualities

This time the Linear Program to solve is:

$$\min_{C,P^U,P^L} \quad \sum_{i\in I,k\in K} C_{ik} + \sum_{j\in J,k\in K} P^L_{jk} + \sum_{j\in J,k\in K} P^U_{jk}$$

$$\text{s.t.} \quad C_{ik} - P^L_{jk} = \Delta^L_{ijk} \; \forall i \in I, j \in J, k \in K$$

$$C_{ik} - P^U_{jk} = \Delta^U_{ijk} \; \forall i \in I, j \in J, k \in K$$

Missing Capacities

- Some problem don't specify the capacity constraint for nodes that have infinite capacity
- We can add unbounded capacity constraints for *orphaned* flow variables
- This will create duplicate nodes, but since the original capacity was infinite, the total capacity won't be affected

Implementation Overview

- The method is implemented in F#, a functional language
- Functional programmers prefer immutable state over mutability
- Powerful list operations are convenient for optimization problems
  - A problem contains a list of constraints
  - A constraint is just a list of `(coefficient, variable)` pairs

Implementation Overview: Why F#?

- F# is statically typed, many bugs are caught at compile time
- Part of the ML-family of programming languages, has named unions and pattern matching
- Can leverage the .NET ecosystem, useful if we want to interface with solvers that provide a C# library
- Async and concurrency support

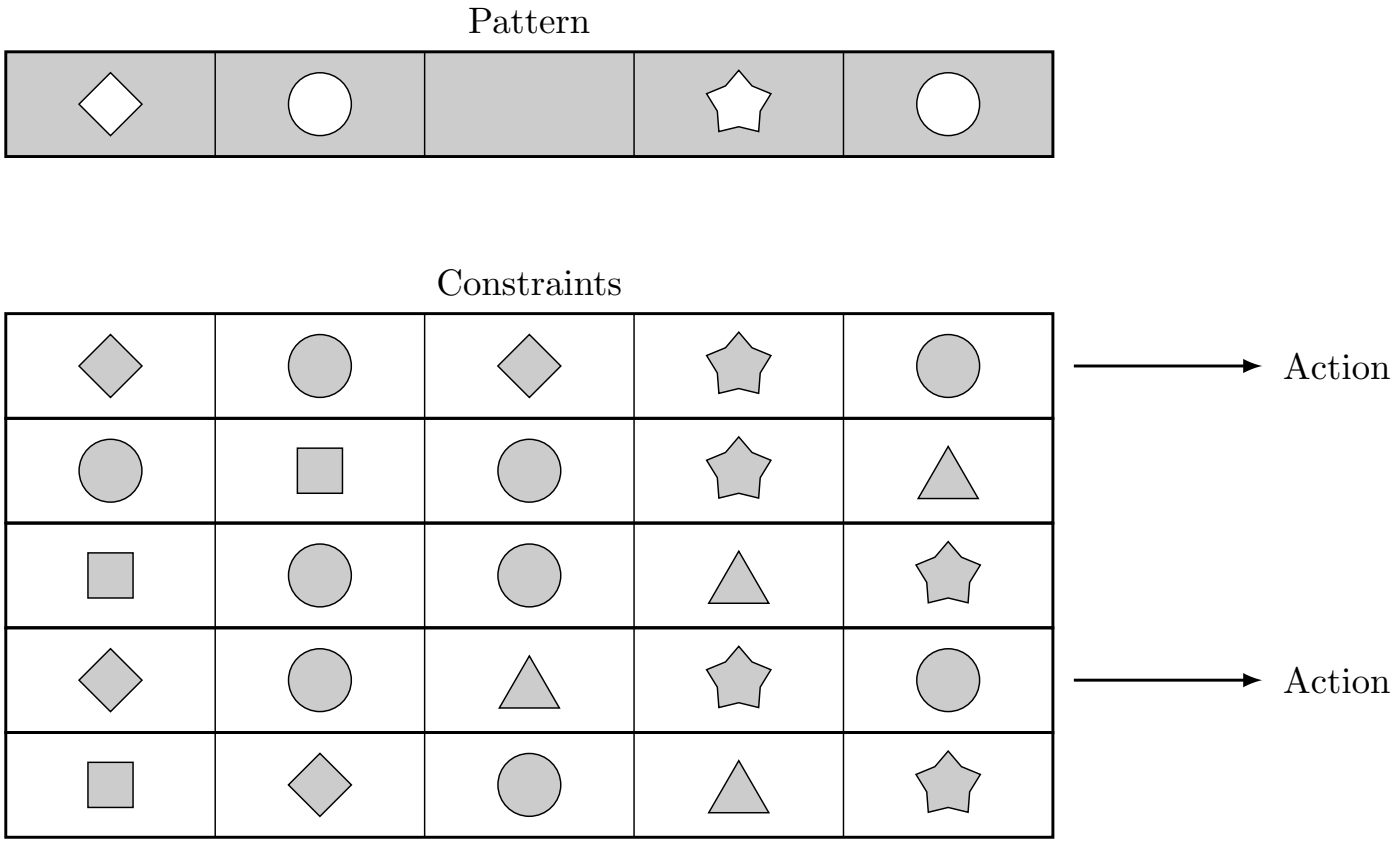
Named Unions

We define all possible type of constraints as:

```
type ConstraintKind =
    | PathDefinitionConstraint
    | ReductionConstraint
    | MaterialBalanceConstraint
    | InputCapacityConstraint of Input
    | PoolCapacityConstraint of Pool
    | OutputCapacityConstraint of Output
    | ProductQualityConstraint of Output
    | OtherConstraint
    | UnidentifiedConstraint
```

Pattern Matching

When a pattern is matched, execute an action



Pattern Matching

In F# we use pattern matching as follows:

```
match constraint with
| InputCapacityConstraint input -> buildInput constraint input
| PoolCapacityConstraint pool -> buildPool constraint pool
| OutputCapacityConstraint output -> buildOutput constraint output
```

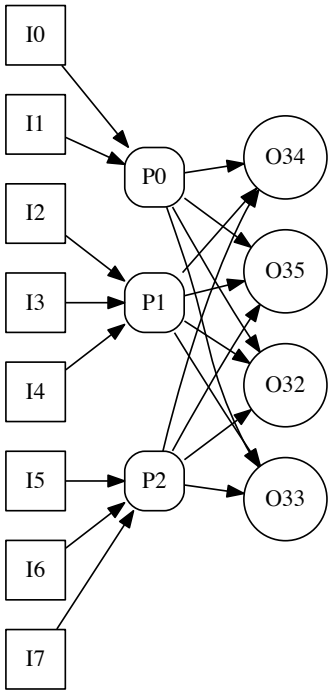
Tested Datasets

The method was tested on the following datasets:

- 70 large-scale standard pooling (Dey and Gupte 2015) examples
- 16 standard and extended (Misener, Thompson, and Floudas 2011) examples
- 1342 MINLPLib2 tests cases

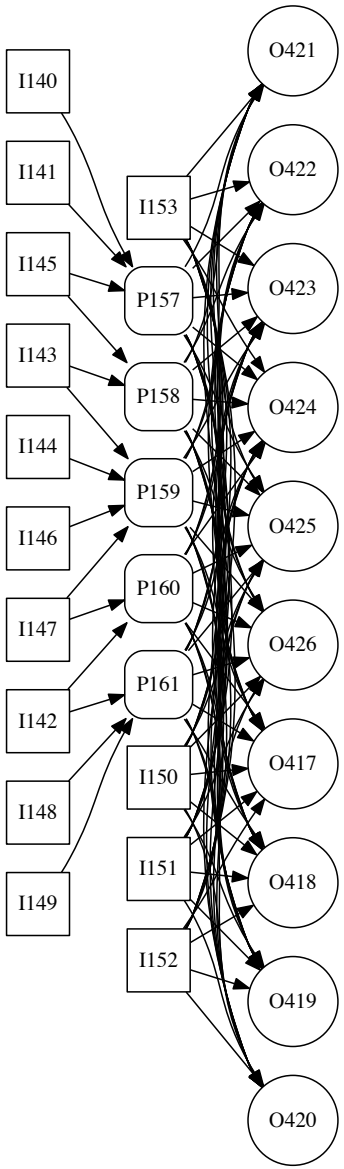
For each, we read a flat optimization problem and output a pooling network.

Example: Adhya



	Adhya 1	Adhya 2	Adhya3
# Constraints	42	50	62
# Variables	13	13	20
# Bilinear Terms	20	20	32
Network Found	Yes	Yes	Yes
Same as Original	Yes	Yes	Yes
Qualities	Yes	Yes	Yes

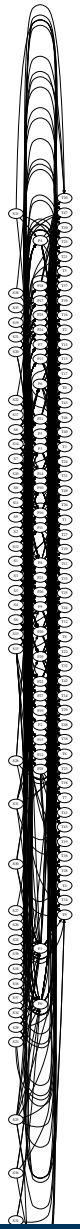
Example: EPA



	EPA Small[1]	EPA Midsize[1]	EPA Large[1]
# Constraints	340	524	1717
# Variables	214	331	1104
# Bilinear Terms	34	69	270
Network Found	Yes	Yes	Yes
Same as Original	Yes	Yes	Yes
Qualities	Yes	Yes	Yes

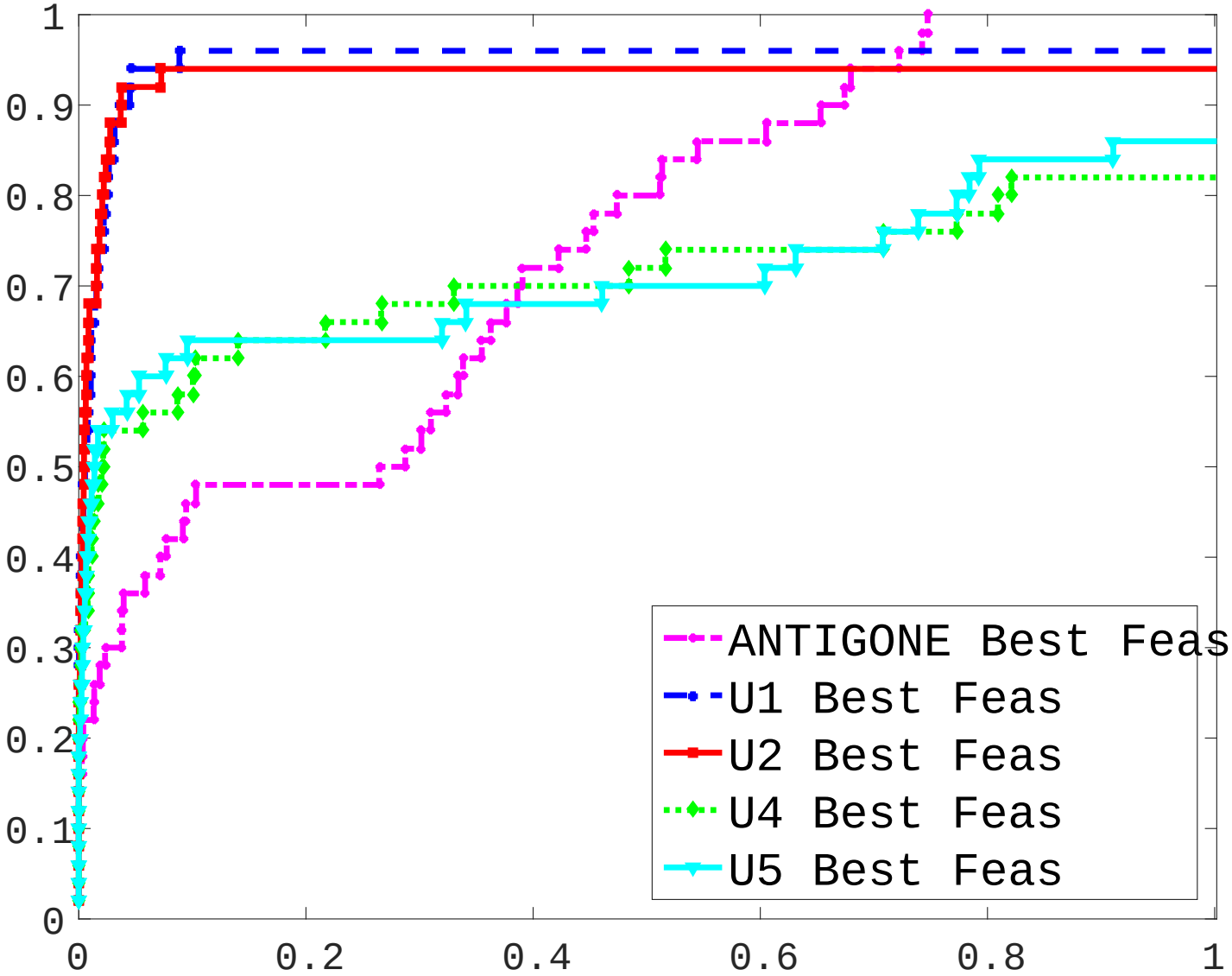
[1] Original problem modified to include output capacities

Example: randstd



	randstd11-20	randstd31-40	randstd51-60
# Constraints	3164	5751	12883
# Variables	2707	4979	11422
# Bilinear Terms	2279	4320	10216
Network Found	Yes	Yes	Yes
Same as Original	Yes	Yes	Yes
Qualities	Yes	Yes	Yes

Performance Profile



Conclusions

- Finding a named structure within a MILP optimization problem allows to use cutting planes and primal heuristics
- We were able to find a pooling network in 6% of MINLPLib2
- After finding the structure, we can apply a good heuristic approach to get a good approximation solution
- Size of the problem is not a limiting factor for the method

Ceccon, Kouyialis, Misener. Using Functional Programming to recognize Named Structure in an Optimization Problem: Application to Pooling. *AIChE Journal*; DOI 10.1002/aic.15308, 2016.

Invited article for special issue honouring Prof. R. W. H. Sargent

Acknowledgments This work is supported by a Royal Academy of Engineering Research Fellowship to R.M. and EPSRC Grant EP/M028240/1.

References I

Achterberg and Raack (2010). "The MCF-separator: Detecting and exploiting multi-commodity flow structures in MIPs". In: *Math Program Comput* 2.2, pp. 125–165.

Alfaki and Haugland (2013). "A multi-commodity flow formulation for the generalized pooling problem". In: *J Glob Optim* 56.3, pp. 917–937.

Baltean-Lugojan and Misener (2016). *A Parametric Approach to the Pooling Problem*. http://www.optimization-online.org/DB_HTML/2016/05/5457.html.

Bixby and Fourer (1988). "Finding Embedded Network Rows in Linear Programs I. Extraction Heuristics". In: *Manag Sci* 34.3, pp. 342–376.

Boland, Kalinowski, and Rigterink (2015). "A polynomially solvable case of the pooling problem". In: *arXiv.org*. scholar: C211DC89-1008-44B7-85B7-47CAB2923759.

Brown and Wright (1984). "Automatic identification of embedded network rows in large-scale optimization models". In: *Math Program* 29.1, pp. 41–56.

D'Ambrosio et al. (2014). *Strong Convex Nonlinear Relaxations of the Pooling Problem*. http://minlp.cheme.cmu.edu/2014/papers/linderoth.pdf.

References II

Dey and Gupte (2015). "Analysis of MILP Techniques for the Pooling Problem". In: *Oper Res* 63.2, pp. 412–427.

Gulpinar et al. (2004). "Extracting pure network submatrices in linear programs using signed graphs". In: *Discrete Appl Math* 137.3, pp. 359–372.

Haugland (2015). "The computational complexity of the pooling problem". In: *J Glob Optim* 64.2, pp. 1–17.

Liberti and Pantelides (2006). "An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms". In: *J Glob Optim* 36.2, pp. 161–189.

Misener, Smadbeck, and Floudas (2015). "Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into GloMIQO 2". In: *Optim Met Softw* 30.1, pp. 215–249.

Misener, Thompson, and Floudas (2011). "APOGEE: Global Optimization of Standard, Generalized, and Extended Pooling Problems via Linear and Logarithmic Partitioning Schemes". In: *Comput Chem Eng* 35.5, pp. 876–892.

Nemhauser, Savelsbergh, and Sigismondi (1994). "MINTO, a Mixed INTeger Optimizer". In: *Oper Res Lett* 15.1, pp. 47–58.

References III

- Roy, Van and Wolsey (1987). "Solving mixed integer programming problems using automatic reformulation". In: *Oper Res* 35.1, pp. 45–57.
- Salvagnin (2016). "Detecting Semantic Groups in MIP models". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*. Lecture Notes in Computer Science.
- Tawarmalani, Ahmed, and Sahinidis (2002). "Product Disaggregation in Global Optimization and Relaxations of Rational Programs". In: *Optim Eng* 3.3, pp. 281–303.
- Tawarmalani and Sahinidis (2002). *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming*. Vol. 65. Theory, Algorithms, Software, and Applications. Boston, MA: Springer Science & Business Media.