

# Advanced MIP Solving with CPLEX

Tobias Achterberg  
CPLEX R&D  
[achterberg@de.ibm.com](mailto:achterberg@de.ibm.com)

# The CPLEX R&D Team



**Tobias Achterberg**  
MIP



**Christian Blik**  
Barrier, (MI)Q(C)P



**John Cui**  
MATLAB



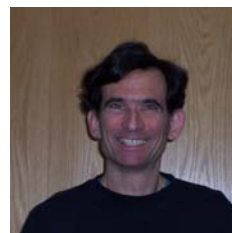
**Emilie Danna**  
MIP



**Mary Fenelon**  
Product Manager



**Daniel Junglas**  
MIP, Excel



**Ed Klotz**  
Product Expert



**Laszlo Ladanyi**  
MIP



**Andrea Lodi**  
MIP



**Robert Schiele**  
Excel, Concert, Tools

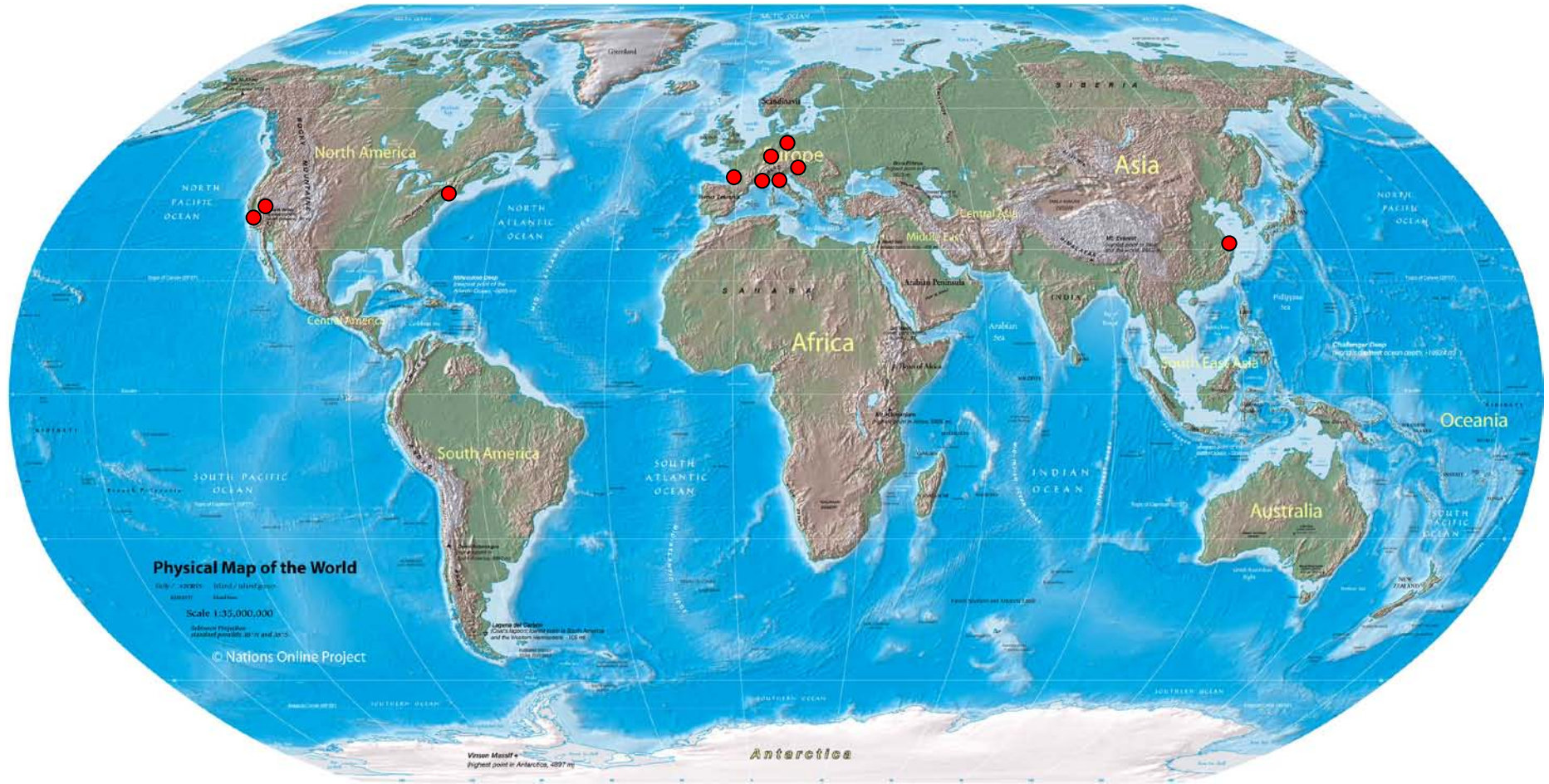


**Philip Starhill**  
Barrier, Python



**Roland Wunderling**  
Simplex, MIP, MATLAB

# The CPLEX Team



# CPLEX Features

- Problem types that can be solved
  - LP: linear programs
  - QP: quadratic programs (convex objective)
  - QCP: quadratically constrained programs (convex constraints)
  - MIP: mixed integer linear program
  - MIQP: mixed integer quadratic program
  - MIQCP: mixed integer quadratically constrained program
- Continuous algorithms (LP, QP, QCP)
  - primal simplex
  - dual simplex
  - barrier

each can solve all three problem types

each can be used as start and sub algorithm in MIP solves

# Solving zib03 with CPLEX Barrier

- experiment conducted by Thorsten Koch (ZIB)
- zib03 is an LP with
  - 29 million columns
  - 20 million rows
  - 104 million non-zeros
  - 3.3 GB just to read in the model
- Solvable with CPLEX parallel barrier
  - 8 threads
  - 256 GB
  - almost



Let's do it...

# CPLEX Features

- Problem types that can be solved
  - LP: linear programs
  - QP: quadratic programs (convex objective)
  - QCP: quadratically constrained programs (convex constraints)
  - MIP: mixed integer linear program
  - MIQP: mixed integer quadratic program
  - MIQCP: mixed integer quadratically constrained program
- Continuous algorithms (LP, QP, QCP)
  - primal simplex
  - dual simplex
  - barrier

each can solve all three problem types

each can be used as start and sub algorithm in MIP solves

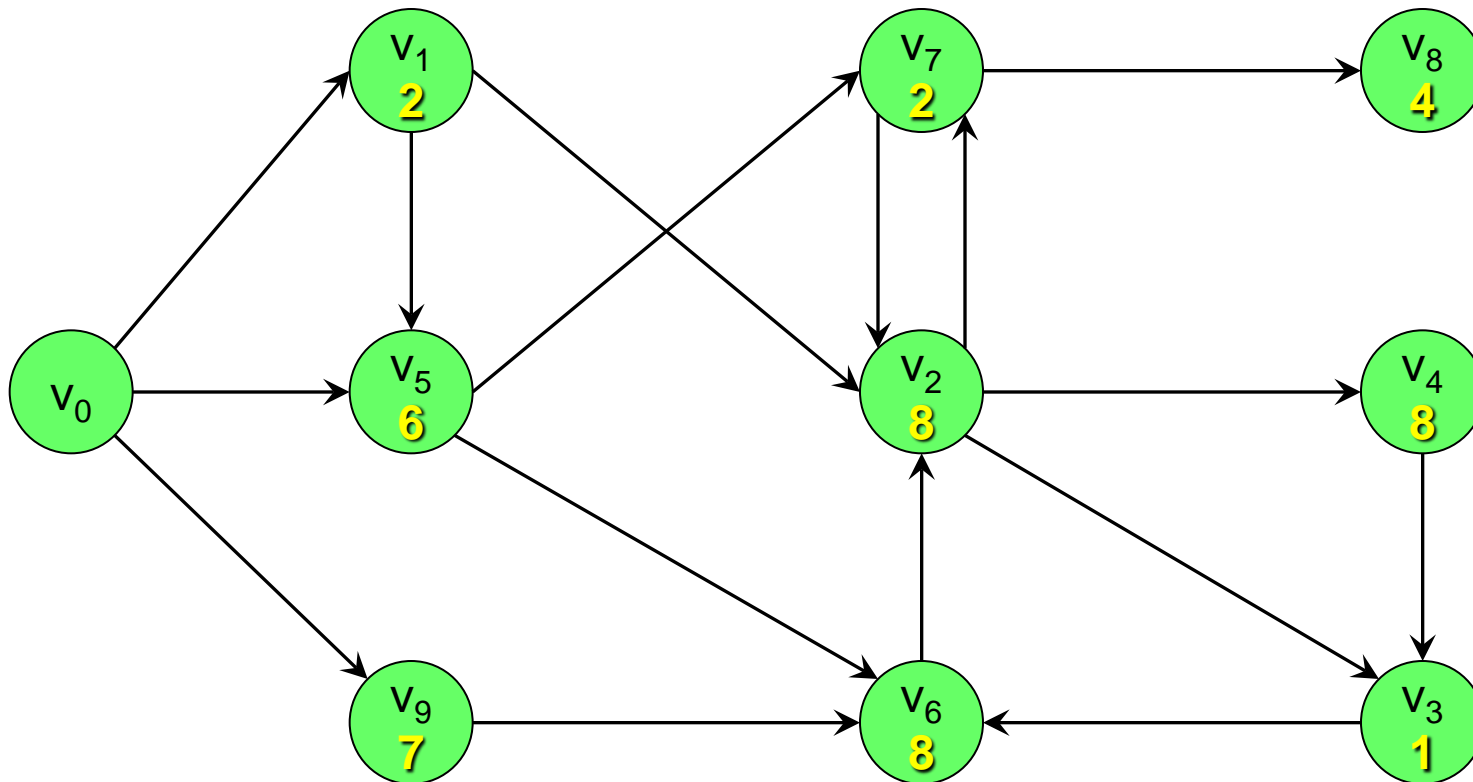
# CPLEX Features

- Control Callbacks
  - MIP callback
    - do something at every node
  - incumbent callback
    - add your own feasibility test
  - heuristic callback
    - add your own primal heuristics
  - cut callback
    - add your own cutting planes
  - branch callback
    - add your own branching rules (including hyperplane branching)
  - node selection callback
    - add your own node selection strategy
  - solve callback
    - add your own relaxation solver
- Try your research ideas inside state-of-the-art environment!

# CPLEX Features

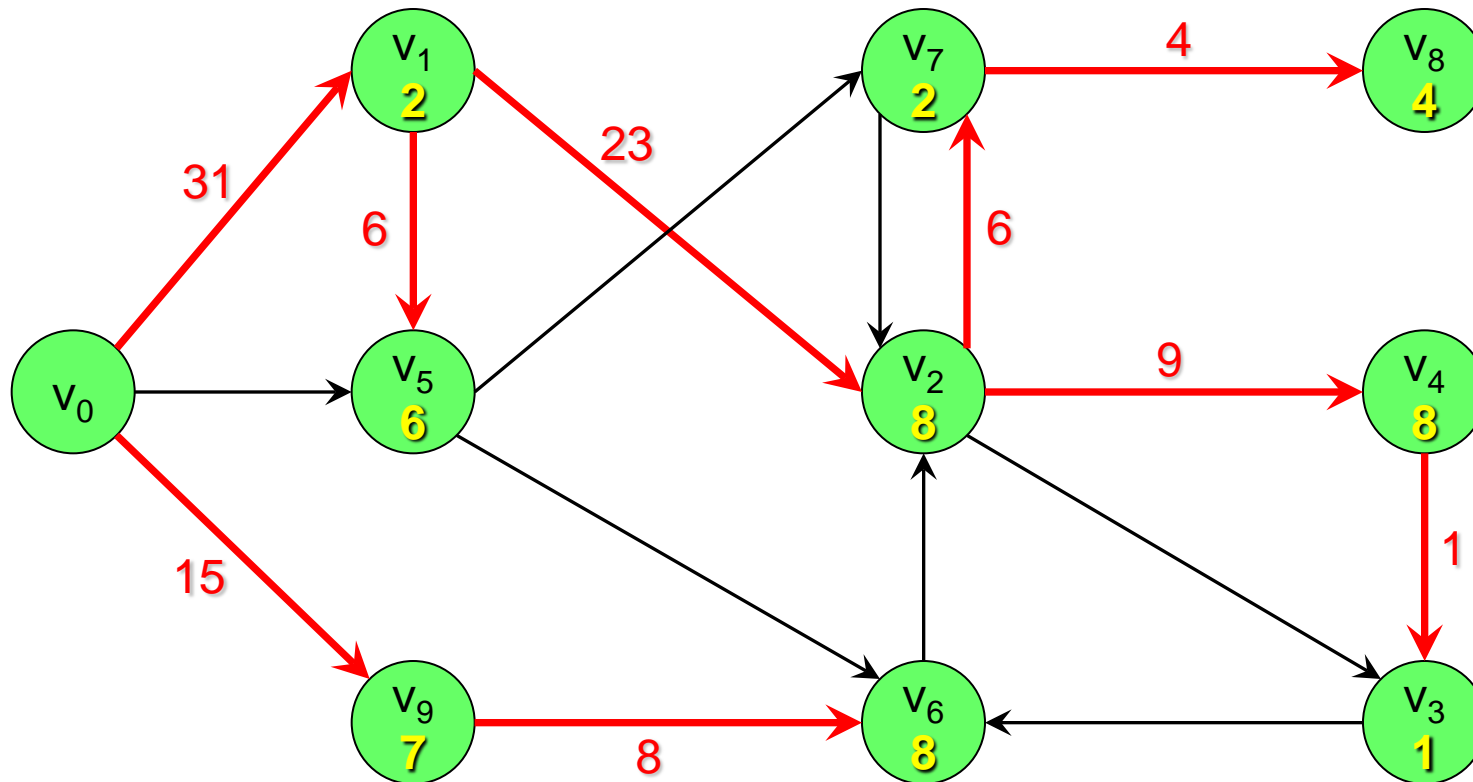
- Tools to improve solving MIP models
  - parameter tuning tool
    - find settings that work best for your problem class
  - polishing
    - improve primal bound
  - solution pool
    - get multiple solutions out of a solve
  - populate
    - get more and diverse solutions
- Tools to analyze MIP model issues
  - FeasOpt (infeasible problems)
    - find minimal relaxation of problem to make it feasible
  - Conflict Refiner (infeasible problems)
    - extract a minimal infeasible subproblem to show the issue
  - Condition Number (numerical issues)
    - estimate the condition number of optimal LP basis

# Arborescence Flow Problem



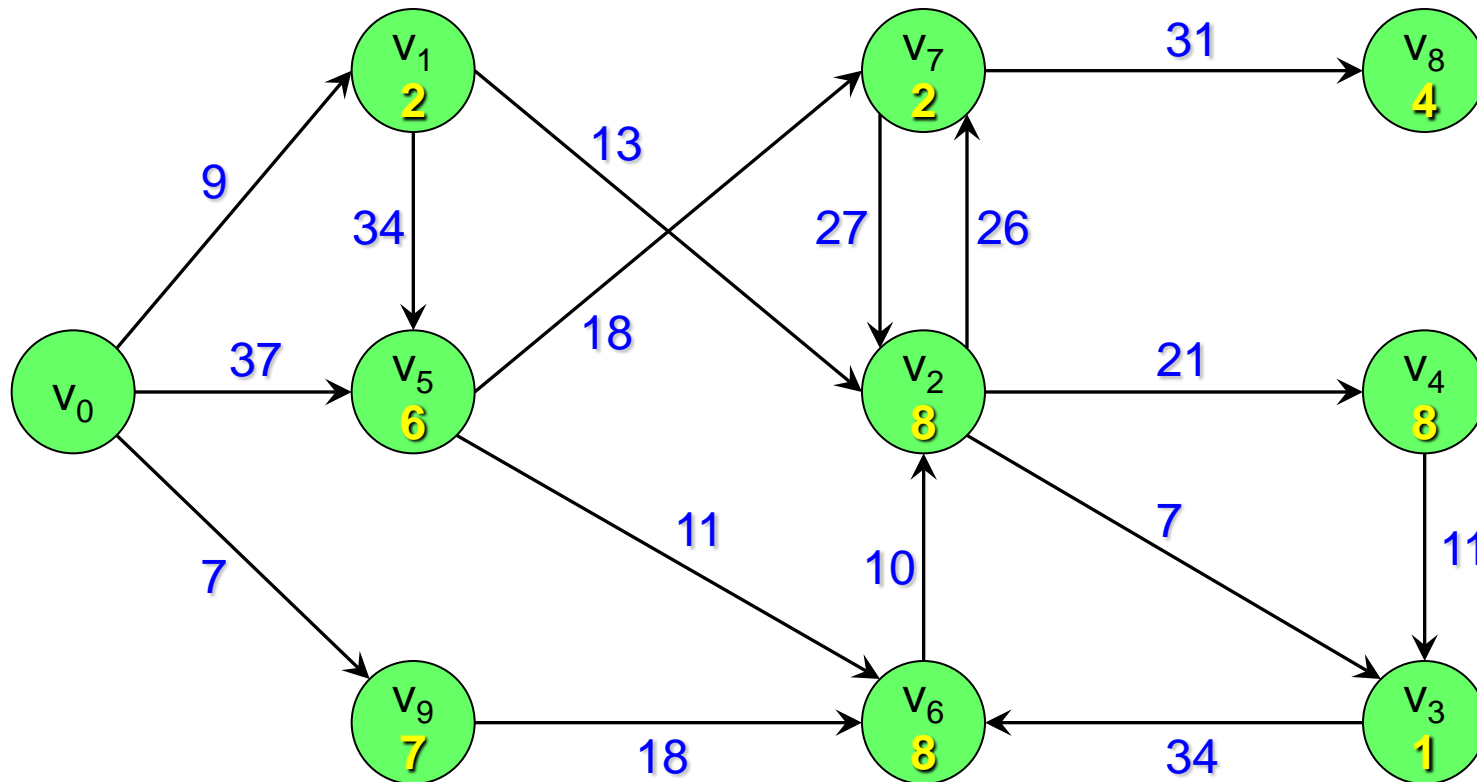
- small version of MIPLIB aflow30a / aflow40b instances
- satisfy **demand** from root node  $v_0$  on unsplittable paths

# Arborescence Flow Problem



- small version of MIPLIB aflow30a / aflow40b instances
- satisfy **demand** from root node  $v_0$  on unsplittable paths
  - result: arborescence rooted at node  $v_0$

# Arborescence Flow Problem

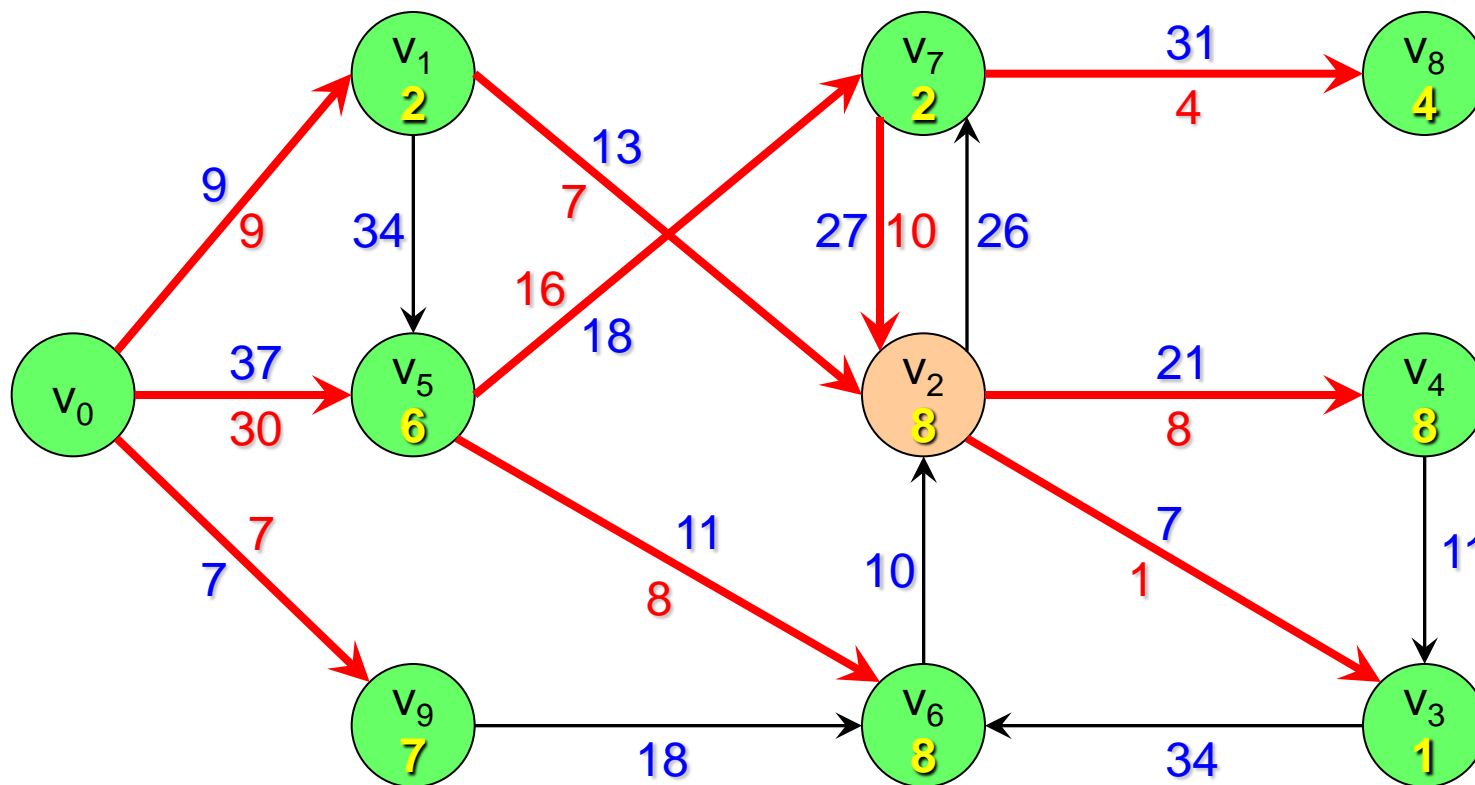


- arborescence flow problem (MPLIB: aflow30a, aflow40b)
- satisfy Let's solve it with CPLEX ... multiple paths
- additional constraints: arc capacities

# Analyzing Infeasible Models

- FeasOpt
  - finds minimal relaxation of constraints for model to become feasible
    - minimize sum of slacks
    - minimize sum of squares of slacks (QP)
    - minimize number of relaxed constraints (MIP)

# Analyzing Infeasible Models – FeasOpt



**Constraint Name**  
artif singleinput\_2

**Slack Value**  
-1.000000\*\*

# Analyzing Infeasible Models

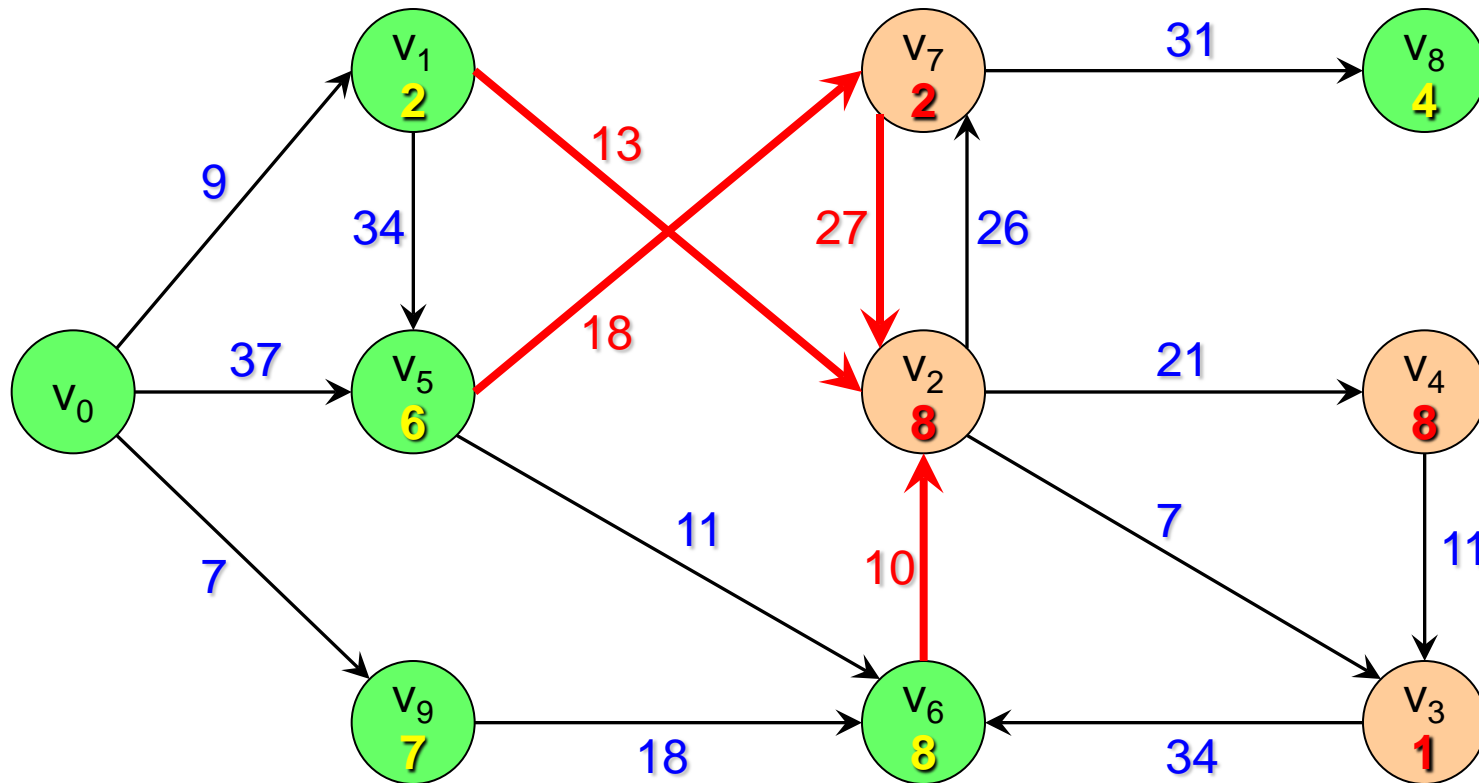
## ■ FeasOpt

- finds minimal relaxation of constraints for model to become feasible
  - minimize sum of slacks
  - minimize sum of squares of slacks (QP)
  - minimize number of relaxed constraints (MIP)

## ■ Conflict Refiner

- identifies (inclusion-wise) minimal set of bounds and constraints that already is infeasible
- small infeasible subproblem points to
  - modeling mistakes
  - incorrect data
  - real-world conflicts in the system being modeled

# Analyzing Infeasible Models – Conflict Refiner



$$\text{singleinput\_2: } x_{1\_2} + x_{6\_2} + x_{7\_2} = 1$$

$$\text{flowequality\_2: } f_{1\_2} + f_{6\_2} + f_{7\_2} - f_{2\_3} - f_{2\_4} - f_{2\_7} = 8$$

$$\text{flowequality\_3: } f_{2\_3} + f_{4\_3} - f_{3\_6} = 1$$

$$\text{flowequality\_4: } f_{2\_4} - f_{4\_3} = 8$$

$$\text{flowequality\_7: } -f_{7\_2} + f_{2\_7} + f_{5\_7} - f_{7\_8} = 2$$

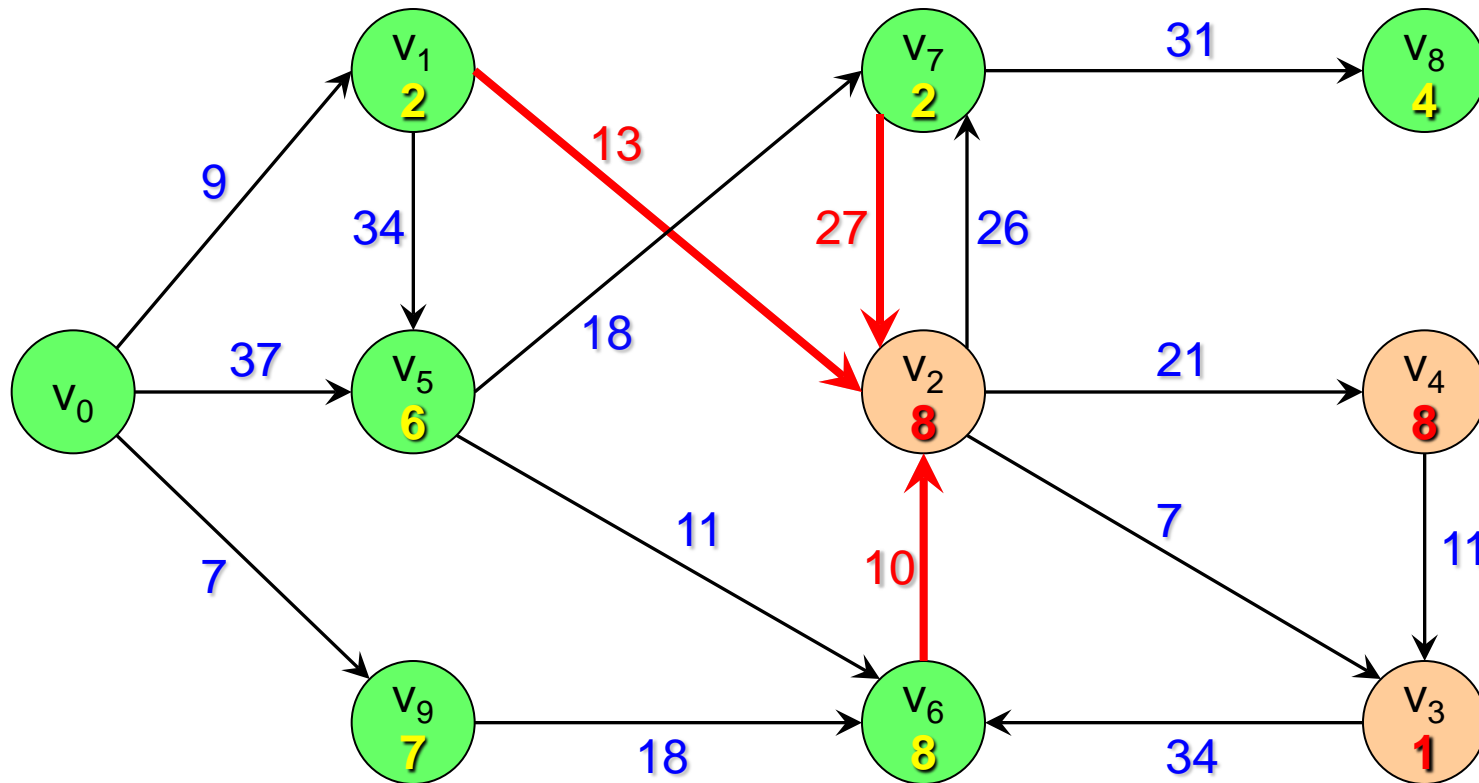
$$\text{capacity\_1\_2: } -13 x_{1\_2} + f_{1\_2} \leq 0$$

$$\text{capacity\_5\_7: } -18 x_{5\_7} + f_{5\_7} \leq 0$$

$$\text{capacity\_6\_2: } -10 x_{6\_2} + f_{6\_2} \leq 0$$

$$\text{capacity\_7\_2: } -27 x_{7\_2} + f_{7\_2} \leq 0$$

# Analyzing Infeasible Models – Conflict Refiner



$$\text{singleinput\_2: } x_{1\_2} + x_{6\_2} + x_{7\_2} = 1$$

$$\text{flowequality\_2: } f_{1\_2} + f_{6\_2} + f_{7\_2} - f_{2\_3} - f_{2\_4} - f_{2\_7} = 8$$

$$\text{flowequality\_3: } f_{2\_3} + f_{4\_3} - f_{3\_6} = 1$$

$$\text{flowequality\_4: } f_{2\_4} - f_{4\_3} = 8$$

$$\text{flowequality\_7: } -f_{7\_2} + f_{2\_7} + f_{5\_7} - f_{7\_8} = 2$$

$$\text{capacity\_1\_2: } -13 x_{1\_2} + f_{1\_2} \leq 0$$

$$\text{capacity\_5\_7: } -18 x_{5\_7} + f_{5\_7} \leq 0$$

$$\text{capacity\_6\_2: } -10 x_{6\_2} + f_{6\_2} \leq 0$$

$$\text{capacity\_7\_2: } -27 x_{7\_2} + f_{7\_2} \leq 0$$



# Use Beyond Infeasible Models

- Conflict Refiner and FeasOpt can do more than fix infeasible problems
  - save time debugging
  - MIP sensitivity analysis
  - MIP performance improvements

# Use Beyond Infeasible Models – Conflict Refiner

## ■ Debugging

- customer said that another optimizer found a better solution than the one CPLEX claimed was optimal
  - fixed the variable bounds at the „better“ solution values
  - conflict revealed that other optimizer had default variable lower bounds of  $-\infty$  instead of 0
- infeasible MIP starts
  - CPLEX 12 allows to apply the conflict refiner to an infeasible MIP start
  - find out which constraints make your „solution“ invalid

# Use Beyond Infeasible Models – Conflict Refiner

- MIP sensitivity analysis
  - given the optimal objective value  $c^*$  of a MIP
  - identify constraints that prevent objective from improving
    - LP: dual solution and reduced costs
  - add constraint  $cx \leq c^* - \epsilon$
  - compute conflict on resulting infeasible MIP
- MIP performance improvements
  - identify irrelevant constraint classes by sensitivity analysis on small instance
  - declare them to be **lazy constraints** in a larger instance

# Use Beyond Infeasible Models – FeasOpt

- Use to find feasible solution
  - feasible solution available if FeasOpt objective = 0
  - use as a MIP start in subsequent mipopt call
    - improvement heuristics like RINS can be applied
- Use to prove infeasibility faster
  - objective function is irrelevant in infeasible models
    - may lead to artificial pseudo costs and inferior branching
  - FeasOpt objective directly models feasibility
    - heuristics, cuts, and branching focus on feasibility
- Tighten MIP formulation
  - Example:
    - soft constraint  $ax + u - v = b$ ,  $x$  integer,  $u, v \geq 0$  penalty variables
    - apply FeasOpt on  $ax = b$  to get min sum of slacks  $f > 0$
    - add valid inequality  $u + v \geq f$

# Numerical Issues

- Feasibility tolerances

- default tolerance is  $10^{-6}$ 
  - 16 decimal digits in double precision
  - assume 10 „correct“ digits, reserve last 6 for round-off errors
  - can be reduced to  $10^{-9}$
- default integrality tolerance is  $10^{-5}$
- definition of tolerances:
  - in the „gray area“ of violation in  $[0, \varepsilon]$ , CPLEX can declare feasibility or infeasibility, whatever it likes better!
  - for marginally infeasible models, both answers are correct!

$$x + 8 \cdot 10^{-7} y = 1 + 9 \cdot 10^{-7}$$

$$x - z = 0.9$$

$$x = 1$$

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

# Numerical Issues

- A harmless model:

$$\begin{array}{ll} \min & x_1 + \dots + x_{400} \\ \text{s.t.} & x_{91} + x_{143} + x_{229} \geq 1 \\ & \dots \\ & x_{33} + x_{42} + x_{93} \geq 1 \\ & -x_1 - x_2 \geq -1 \\ & \dots \\ & -x_{399} - x_{400} \geq -1 \\ & x_j \in \{0,1\} \text{ for all } j = 1, \dots, 400 \end{array}$$

- 400 binary variables
- 520 constraints
- 1359 non-zeros, all +/-1

# Condition Number

- CPLEX solves systems of form:
- exact solution is:
- errors in b give:

$$Bx = b$$

$$x = B^{-1}b$$

$$x + \delta x = B^{-1}(b + \delta b)$$

$$\Rightarrow \delta x = B^{-1}\delta b$$

- Cauchy-Schwarz inequality:
- Cauchy-Schwarz for original system:
- Combine and rearrange:

$$\|\delta x\| \leq \|B^{-1}\| \cdot \|\delta b\|$$

$$\|b\| \leq \|B\| \cdot \|x\|$$

$$\frac{\|\delta x\|}{\|x\|} \leq \|B\| \cdot \|B^{-1}\| \cdot \frac{\|\delta b\|}{\|b\|}$$

# Condition Number

- Condition number of B is defined as  $\kappa(B) = \|B\| \cdot \|B^{-1}\|$

$$\|\delta x\|/\|x\| \leq \kappa(B) \cdot \underbrace{\|\delta b\|/\|b\|}_{\text{machine precision} \approx 10^{-16}}$$

machine precision  $\approx 10^{-16}$

- CPLEX feasibility tolerance =  $10^{-6}$
- Need error less than tolerances  
 $\Rightarrow \kappa(B) \leq 10^{10}$  is a pertinent threshold for default settings
- CPLEX calculates approximate condition number
  - in terms of infinity norm

# Condition Number

- CPLEX provides condition number of optimal basis  $B$ 
  - Large condition number indicates that numerical issues are likely to occur
  - Low condition number does not necessarily imply numerical stability
- CPLEX encounters many different bases while solving an LP or MIP!
  - root node LP is not necessarily the worst troublesome

# Conclusions so far...

- Additional tools in CPLEX help during modeling phase
  - FeasOpt and Conflict Refiner can point you to
    - errors in model
    - errors in data
    - real-world conflict
  - FeasOpt and Conflict Refiner can help you to
    - understand your model better
    - derive valid inequalities to tighten the MIP formulation
    - perform MIP sensitivity analysis
  - the condition number and solution quality output can identify numerical issues in your model

# New Ingredients in CPLEX 12


- Connectors to Excel, MATLAB, Python
- Pseudocost updates for infeasible nodes
- Multi-commodity-flow cuts
- A lot of additional improvements

# Pseudocosts

$$\bigcirc \underline{c} = 2$$

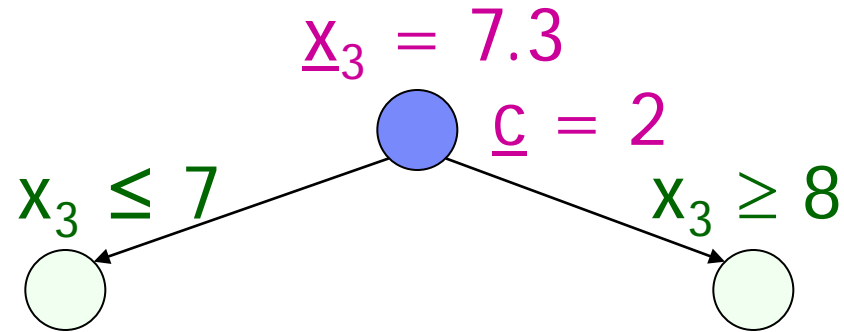
- LP relaxation yields **lower bound**

# Pseudocosts

$$\underline{x}_3 = 7.3$$

$$\underline{c} = 2$$

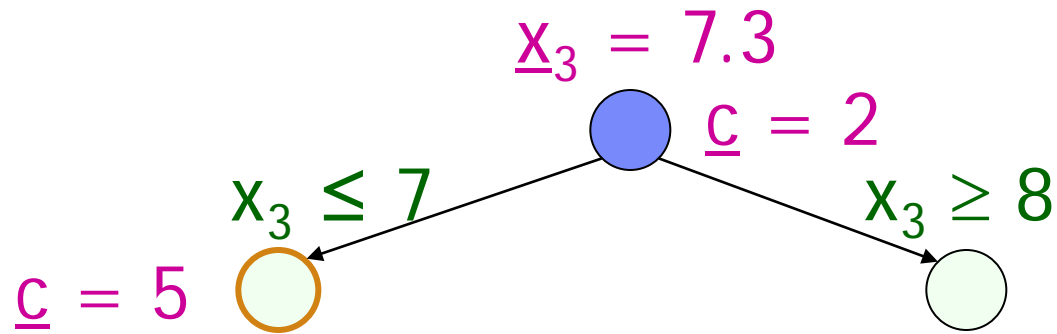
- LP relaxation yields **lower bound**
- integer variable has fractional LP value

# Pseudocosts



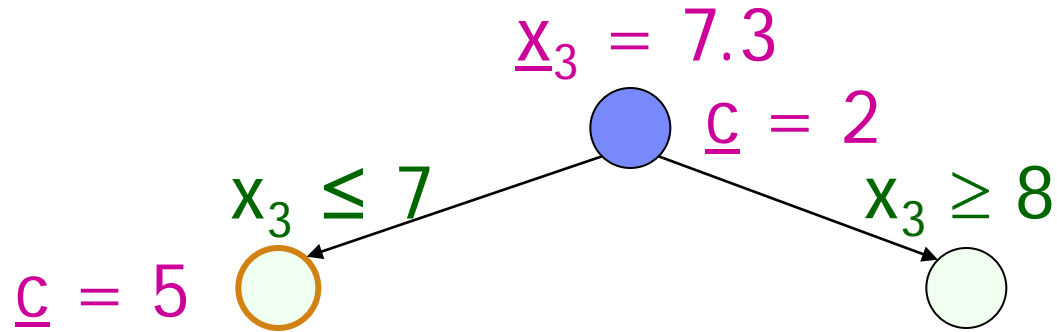
- LP relaxation yields **lower bound**
- integer variable has fractional LP value
- branching decomposes problem into subproblems

# Pseudocosts



- LP relaxation yields **lower bound**
- integer variable has fractional LP value
- branching decomposes problem into subproblems
- LP relaxation is solved for subproblems

# Pseudocosts

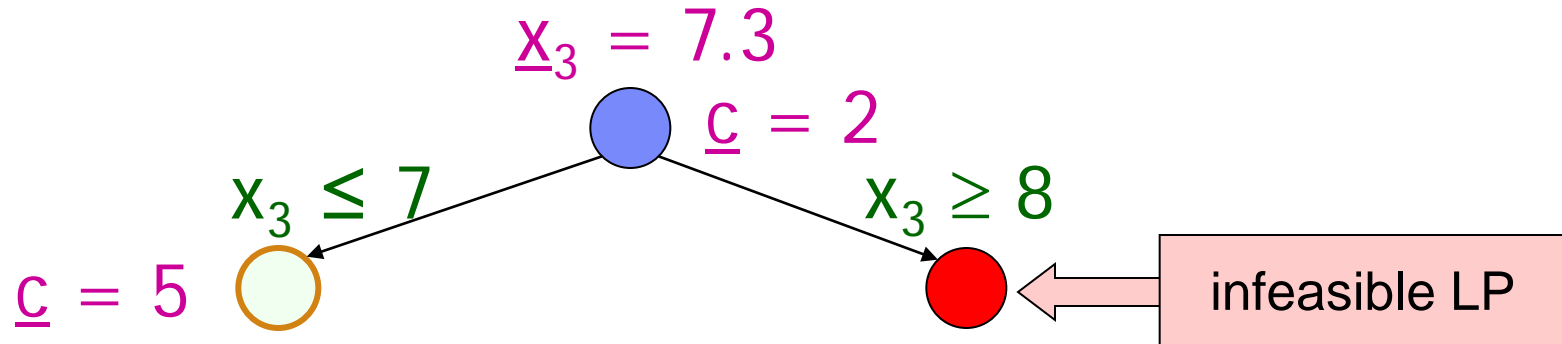


- history of objective changes caused by branching on specific variable

- objective gain per unit: 
$$\zeta_3^- = \frac{5-2}{7.3-7} = \frac{3}{0.3} = 10$$

- down/upwards pseudo costs  $\Psi_j^-$ ,  $\Psi_j^+$ :  
average of all objective gains per unit

# Pseudocost Updates for Infeasible Nodes



- history of objective changes caused by branching on specific variable

- objective gain per unit: 
$$\zeta_3^+ = \frac{\infty - 2}{8 - 7.3} = \frac{\infty}{0.7} = \infty$$

- down/upwards pseudo costs  $\Psi_j^-$ ,  $\Psi_j^+$ :  
average of all objective gains per unit

# Pseudocost Updates for Infeasible Nodes

- How to update pseudocosts if a node is infeasible?
  - CPLEX 11 strategy: just ignore those nodes
- New CPLEX 12 strategy
  - associate infeasible nodes with a fake objective value
    - compute the fake objective value from the average pseudocost recorded on feasible nodes

$$\zeta_3^+ = \frac{c^F - 2}{8 - 7.3} = \frac{c^F - 2}{0.7}$$

$$c^F = 2 + (0.7 \cdot \bar{\Psi}^+ + \varepsilon) \cdot M$$

The diagram illustrates the calculation of the fake objective value  $c^F$ . It shows the equation  $c^F = 2 + (0.7 \cdot \bar{\Psi}^+ + \varepsilon) \cdot M$ . A box labeled "average over all j" points to  $\bar{\Psi}^+$ . A box labeled "big number" points to  $M$ . A box labeled "small number" points to  $\varepsilon$ .

- ignore infeasible nodes after first feasible has been found

# Computational Results – Regular Models

- Time to optimality
  - 2105 models
  - solved to optimality in 0.1 second to 10000 seconds
  - overall improvement: 0% to 2%

# Computational Results – Regular Models

- **Time to first integer solution**
  - 54 models where CPLEX does not find a solution at the root
  - hard models:
    - solved to optimality in more than 1000 seconds
    - most often in more than 10k seconds
  - **Improvement in time: 1.47x faster**
  - Same number of solutions found
  - Quality of first solution slightly improved
    - 32 ties, 15 improvements, 7 degradations

# Computation Results – Satisfiability Models

- 12 models without objective and that require branching to be solved
- Randomly permuted 10 times each (yields 120 models)
- **Time to optimality: 3x faster**
- Performance variability: -36%

- 222 models obtained from regular models by deleting the objective, and that require branching to be solved

Original set	# of models	Speedup
mip0	26	1.00
mip1	25	1.00
mip10	44	1.27
mip100	64	1.69
mip1000	30	2.13
mip10000	33	2.56

# Computational Results – Other Special Models

- 25 infeasible models that require branching to be solved
  - Randomly permuted 10 times each
  - Time to prove infeasibility: 20% faster
  - Performance variability: -11%
  
- 17 models where feasibility is the main issue and that require branching to be solved
  - Randomly permuted 10 times each
  - Time to optimality: 33% faster
  - Performance variability: -32%

# Pseudocost Updates for Infeasible Nodes – Summary

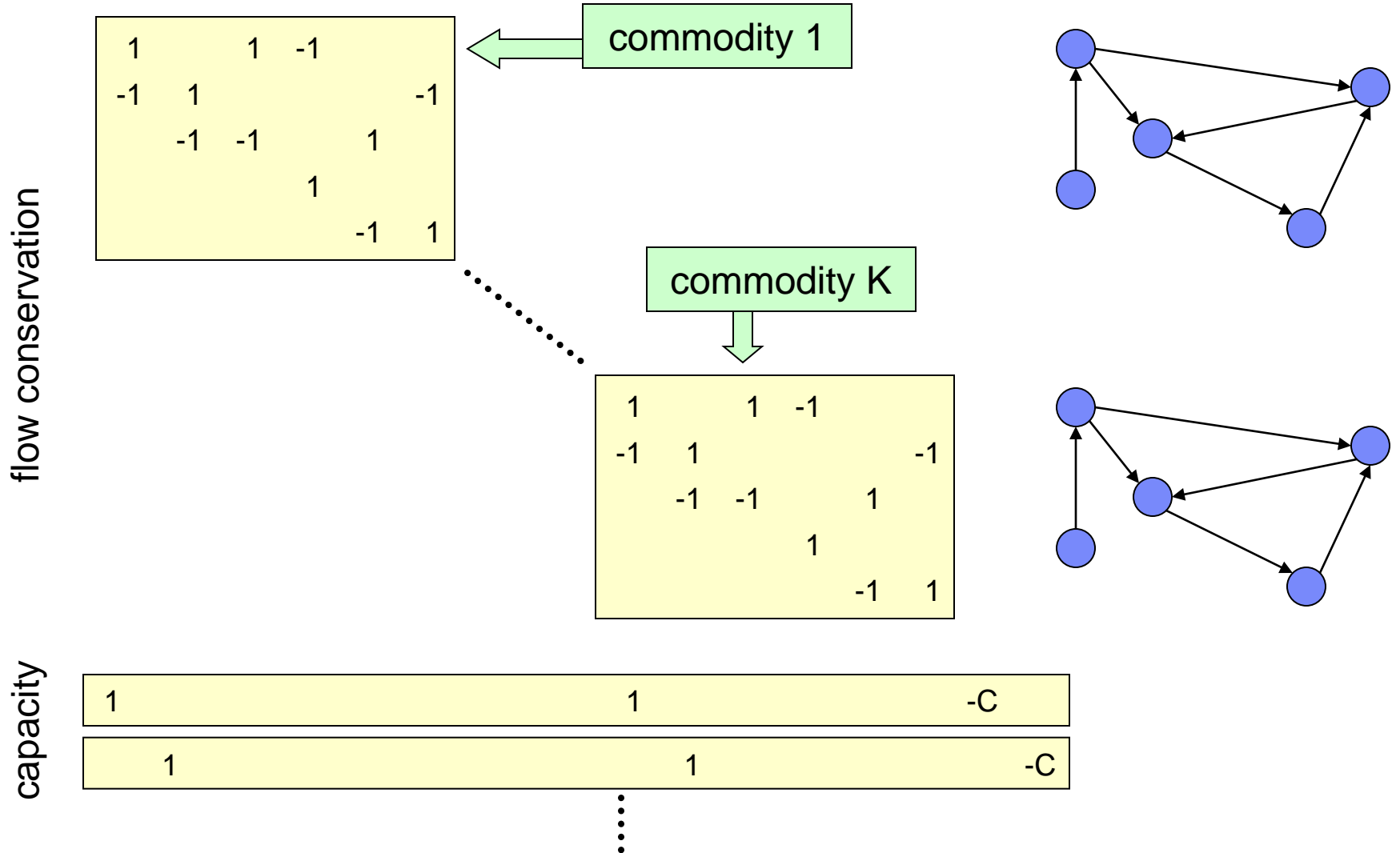
- How to update pseudocosts if a node is infeasible?
  - CPLEX 11 strategy: just ignore those nodes
- New CPLEX 12 strategy
  - associate infeasible nodes with a fake objective value
  - ignore infeasible nodes after first feasible has been found
- Results
  - improvement in time to optimality for satisfiability models and other special models
  - improvement in time to first solution for regular models
  - small improvement in time to optimality for regular models



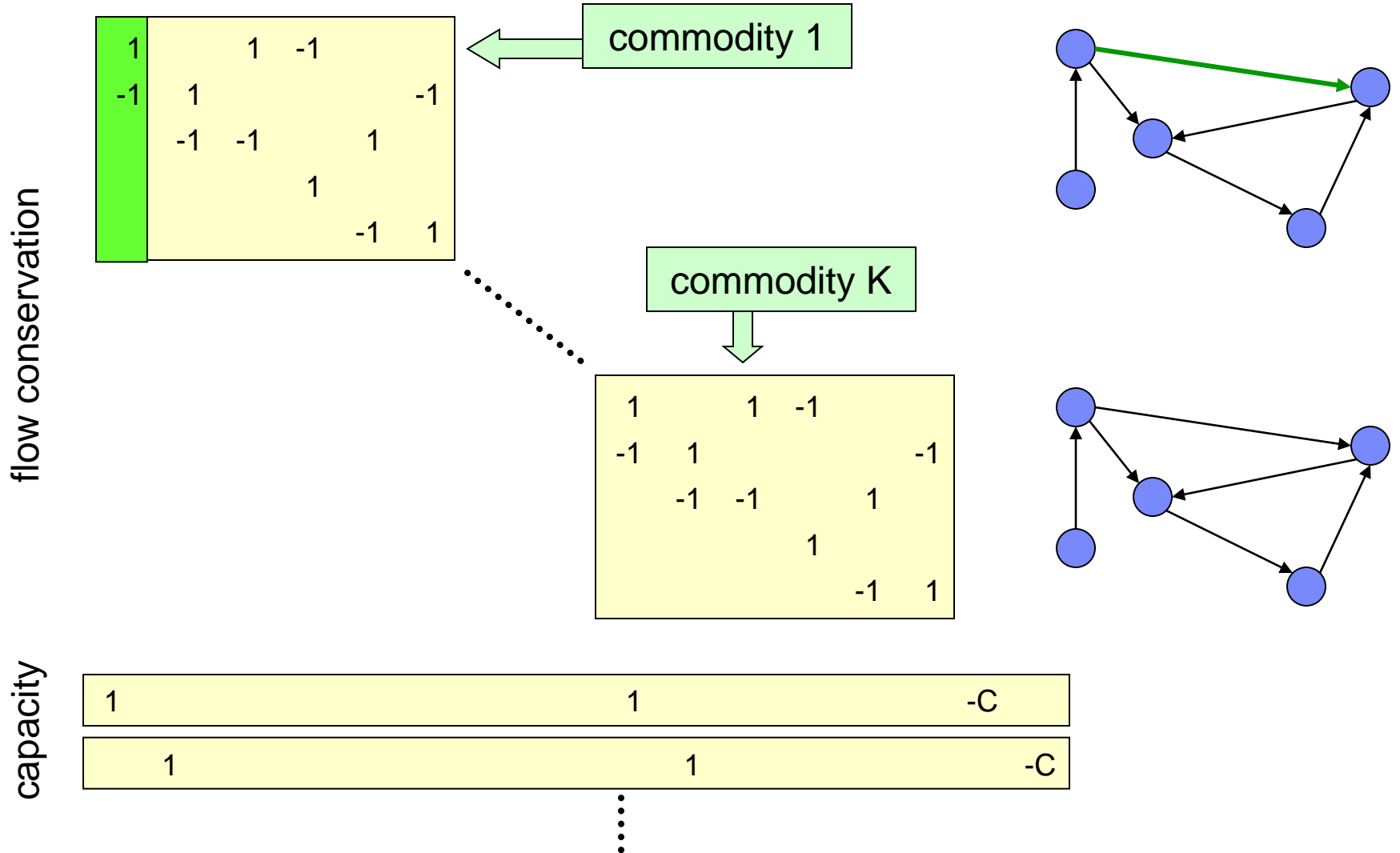
# Multi-Commodity-Flow Cuts

- Observations:
  - Capacitated network flow is a common substructure of MIP models
  - Numerous papers deal with separating cuts for network design problems
- Idea:
  - automatically identify network structure in general MIPs
  - apply cut separation from network design literature
- Goals:
  - should help for models that contain networks
  - **should not degrade performance on other models**

# Multi-Commodity Fixed Charge Network Flow



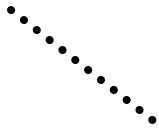
# Multi-Commodity Fixed Charge Network Flow



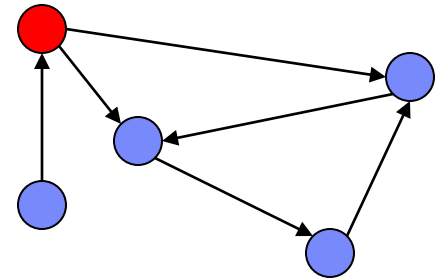
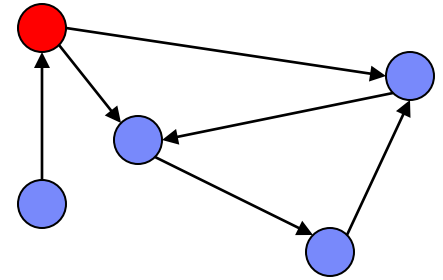
# Multi-Commodity Fixed Charge Network Flow

flow conservation

1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
				-1	1



1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
				-1	1



capacity

1			1			-C
---	--	--	---	--	--	----

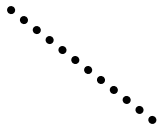
	1			1			-C
--	---	--	--	---	--	--	----



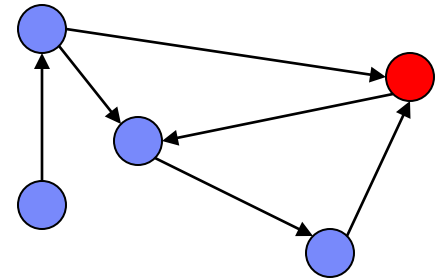
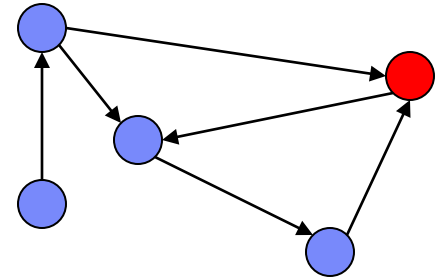
# Multi-Commodity Fixed Charge Network Flow

flow conservation

1		1	-1		
-1	1			-1	
	-1	-1		1	
			1		
				-1	1



1		1	-1		
-1	1			-1	
	-1	-1		1	
			1		
				-1	1



capacity

1			1			-C
---	--	--	---	--	--	----

1			1			-C
---	--	--	---	--	--	----



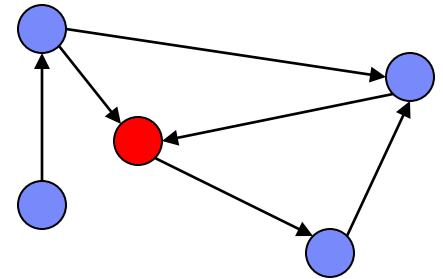
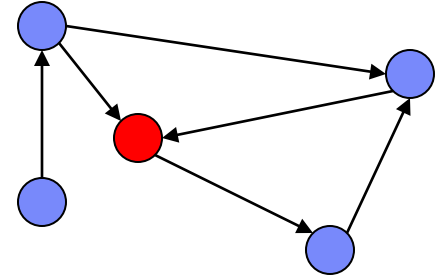
# Multi-Commodity Fixed Charge Network Flow

flow conservation

1		1	-1	
-1	1			-1
-1	-1		1	
		1		
			-1	1



1		1	-1	
-1	1			-1
-1	-1		1	
		1		
			-1	1



capacity

1			1			-C
---	--	--	---	--	--	----

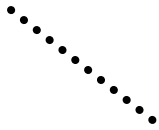
	1			1		-C
--	---	--	--	---	--	----



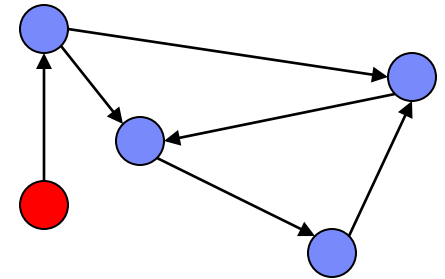
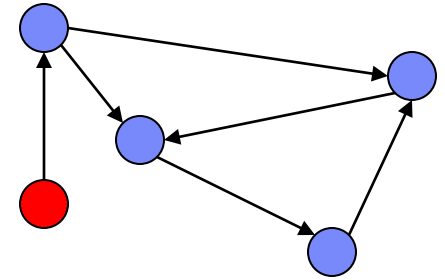
# Multi-Commodity Fixed Charge Network Flow

flow conservation

1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
			-1	1	



1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
			-1	1	



capacity

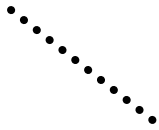
1			1			-C
	1				1	-C



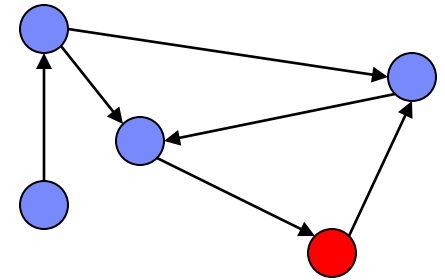
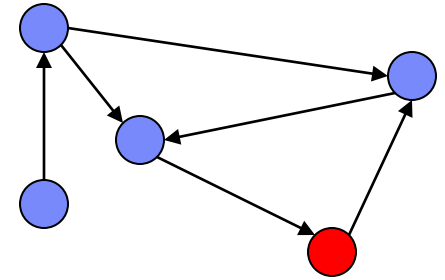
# Multi-Commodity Fixed Charge Network Flow

flow conservation

1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
				-1	1



1		1	-1		
-1	1				-1
	-1	-1		1	
			1		
				-1	1



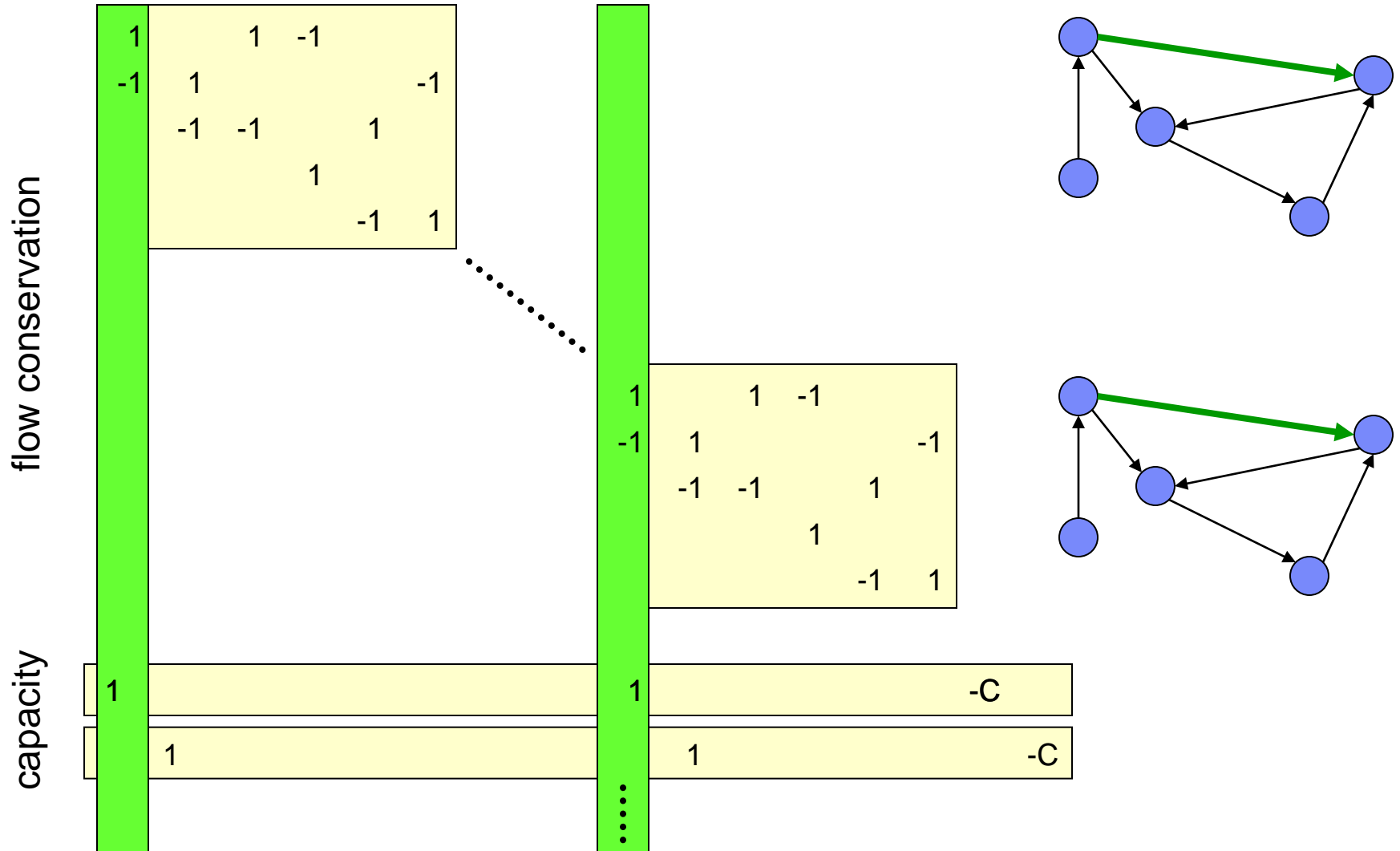
capacity

1			1			-C
---	--	--	---	--	--	----

1			1			-C
---	--	--	---	--	--	----

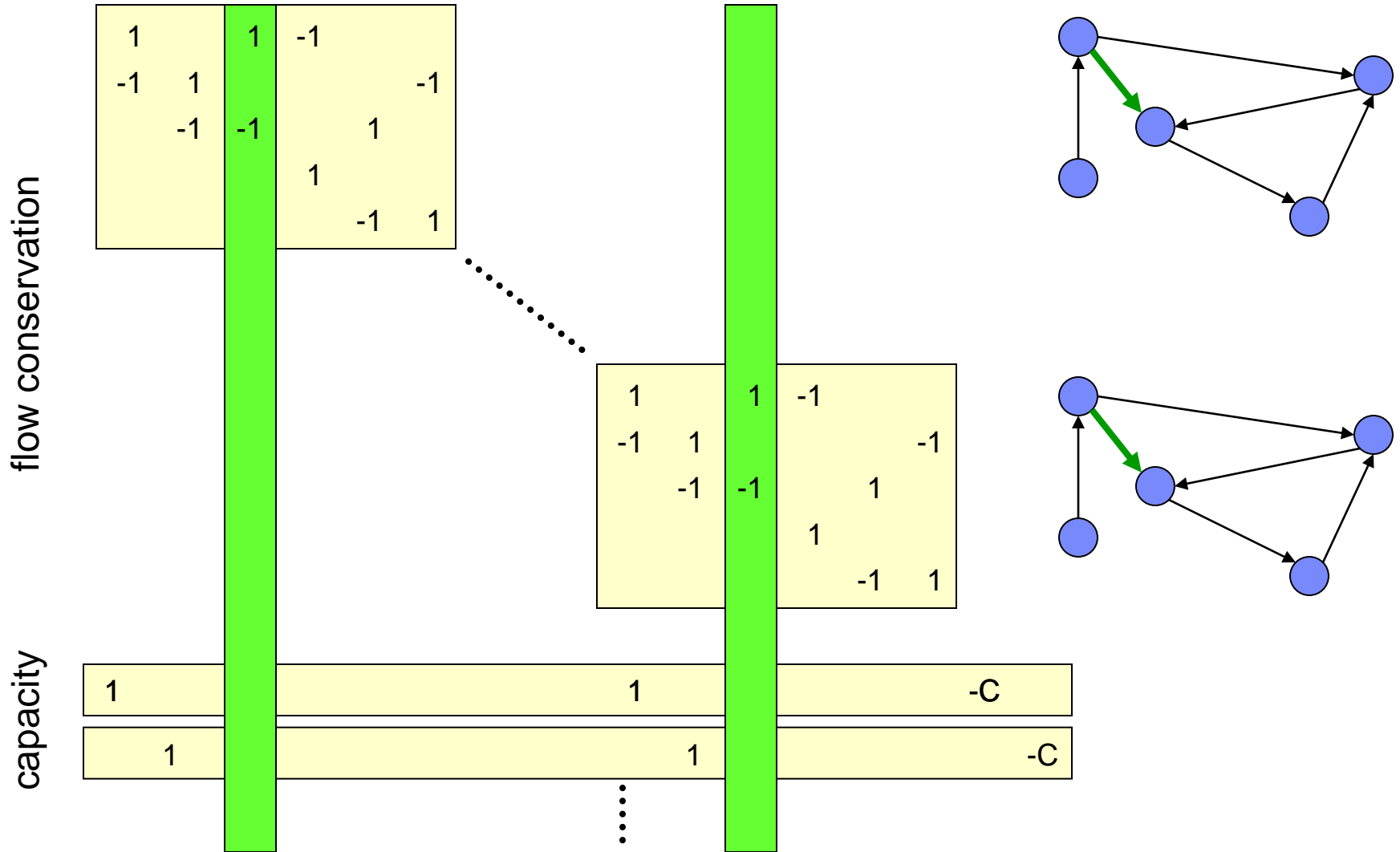


# Multi-Commodity Fixed Charge Network Flow

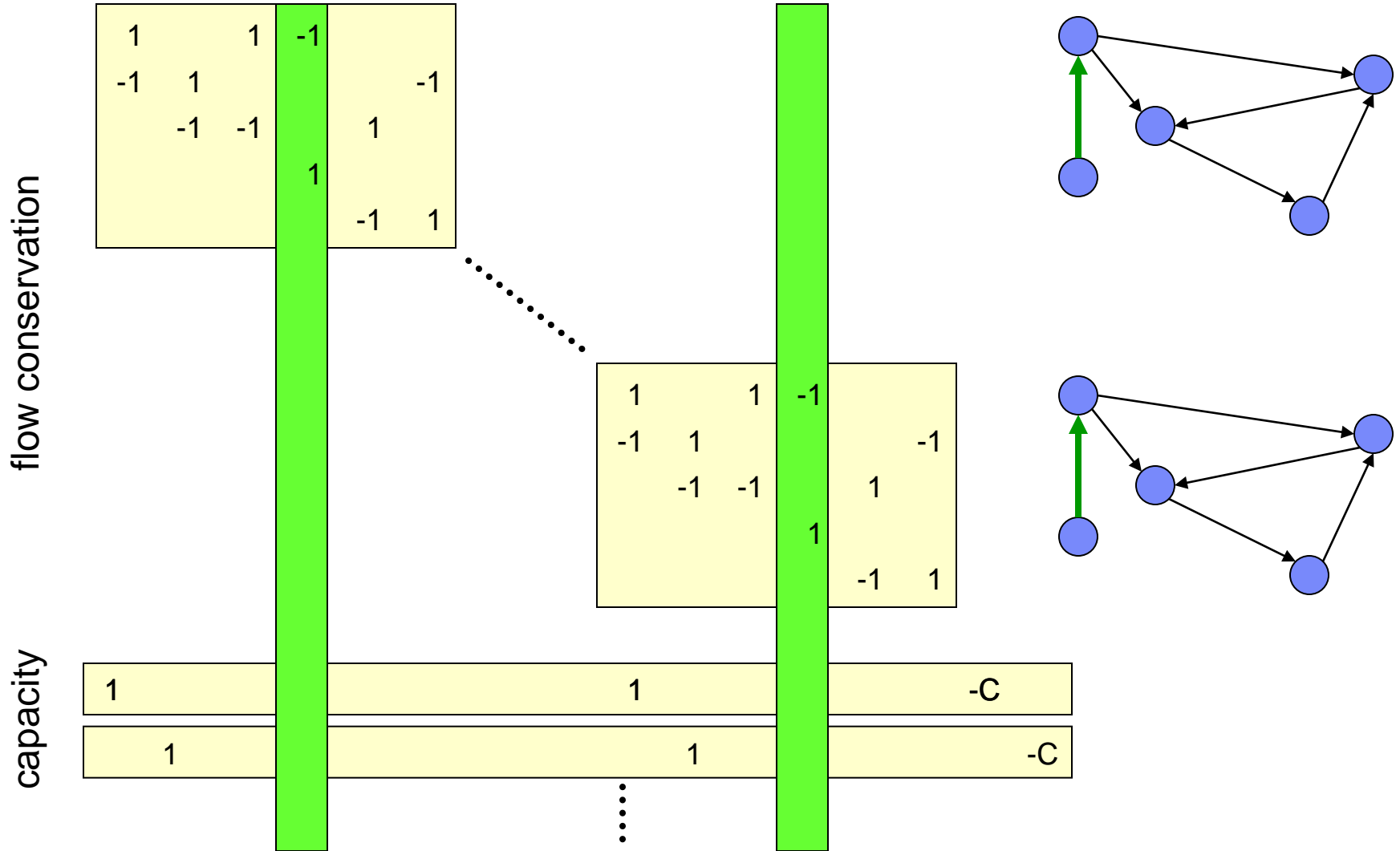




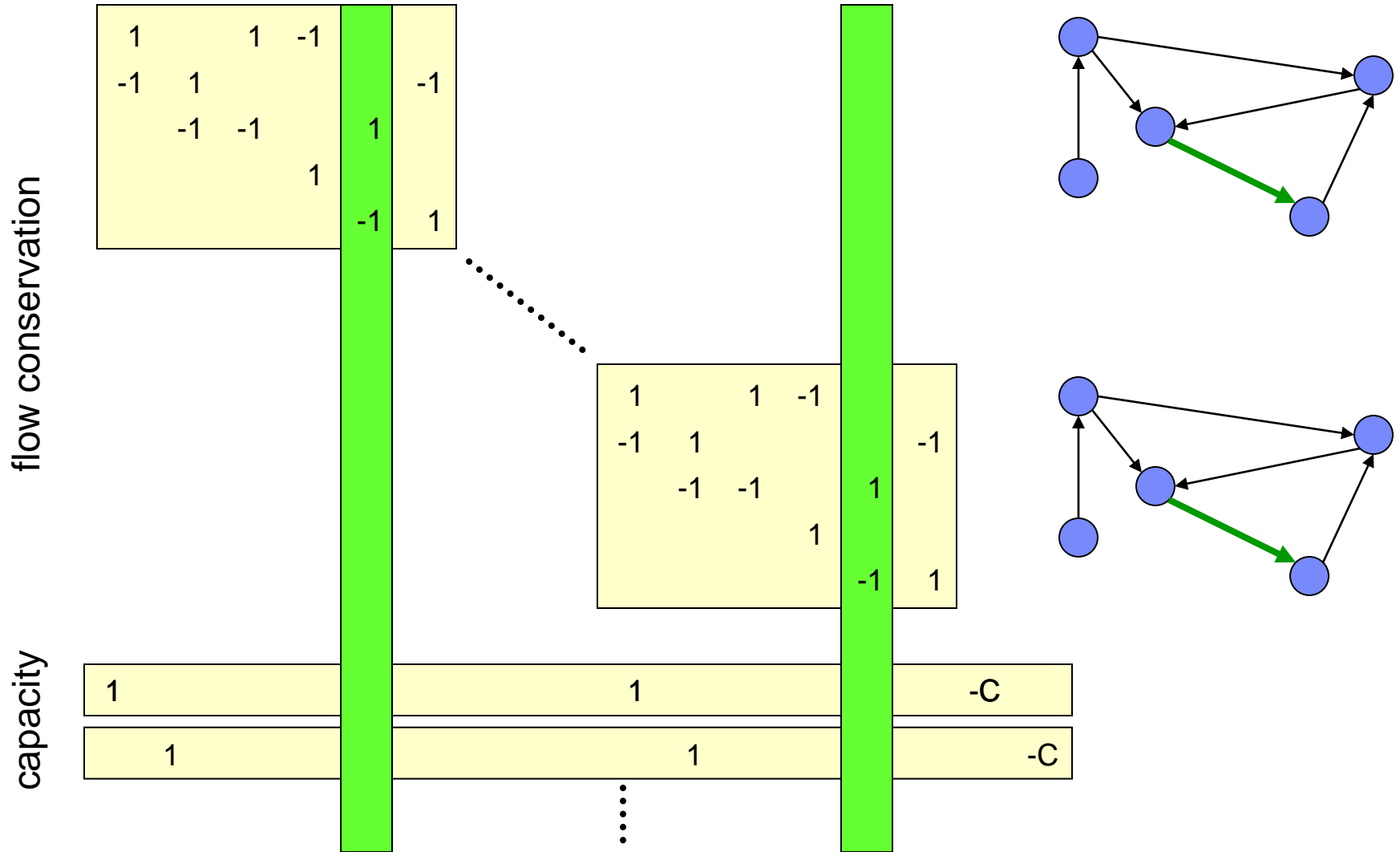
# Multi-Commodity Fixed Charge Network Flow



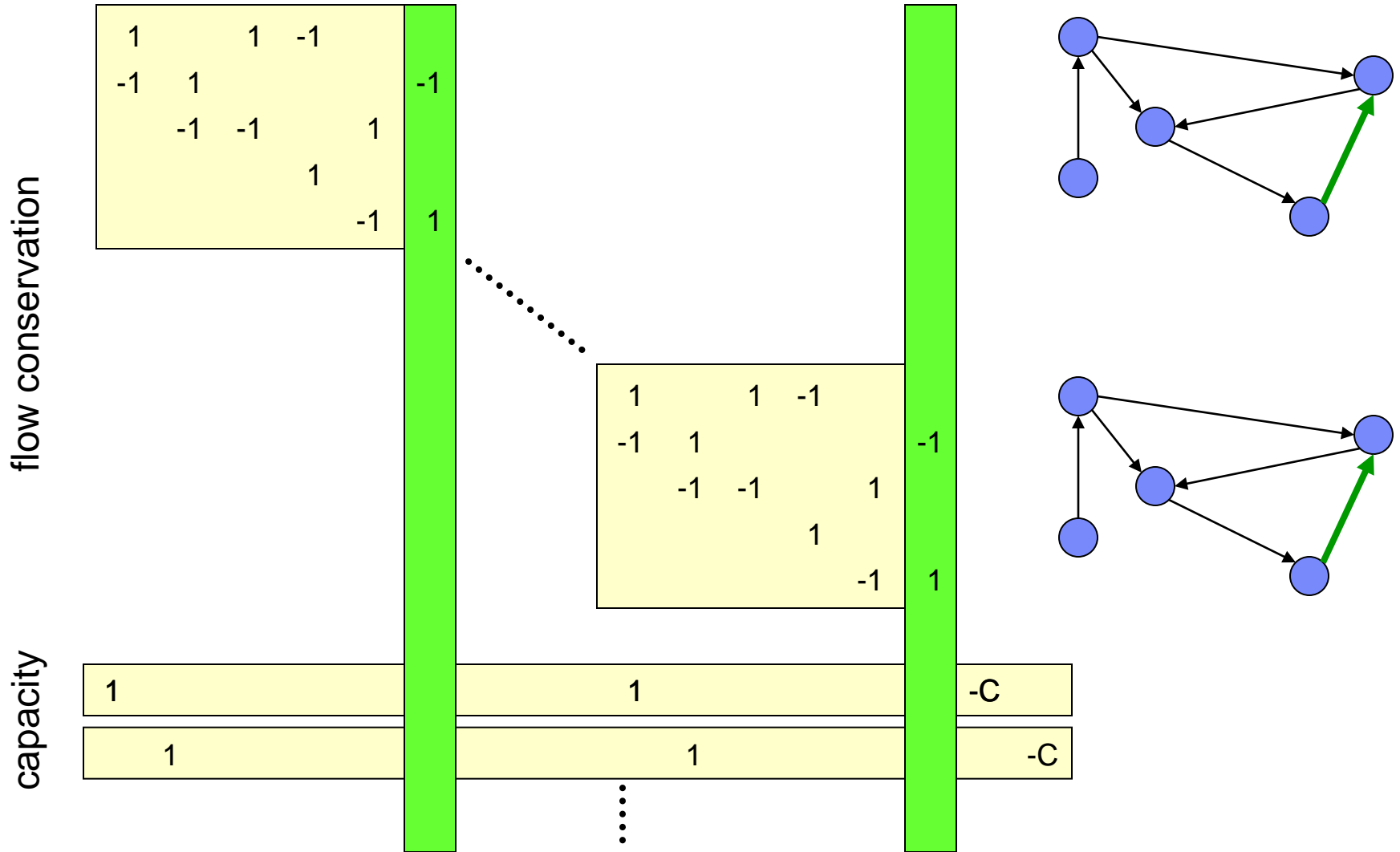
# Multi-Commodity Fixed Charge Network Flow



# Multi-Commodity Fixed Charge Network Flow



# Multi-Commodity Fixed Charge Network Flow

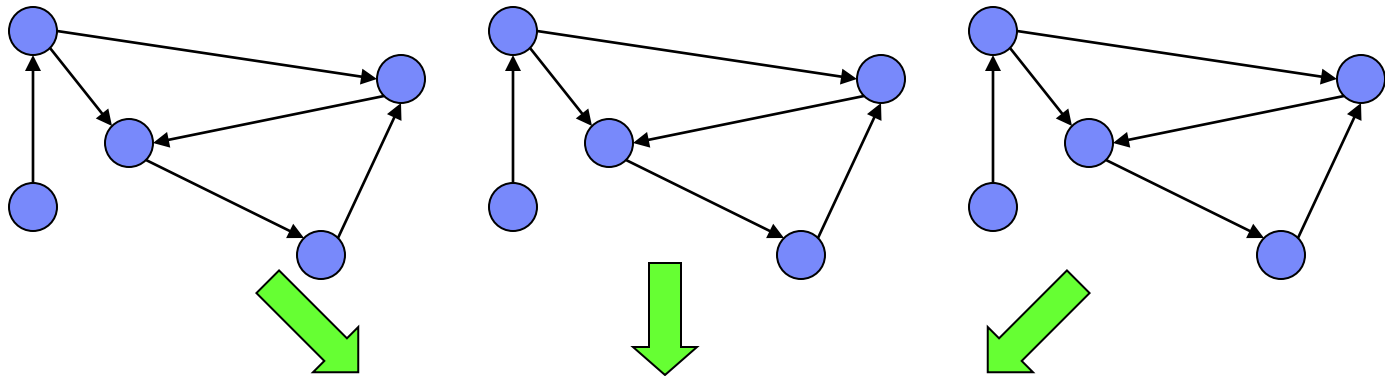


# Multi-Commodity-Flow Cuts – Network Detection

## 1. Detect flow structure

- rows with +/-1 coefficients
- at most one +1 and one -1 per column
- result:
  - directed graph, one component per commodity

1		1	-1		
-1	1				-1
		-1	-1		1
				1	
					-1
					1



graph isomorphism problem

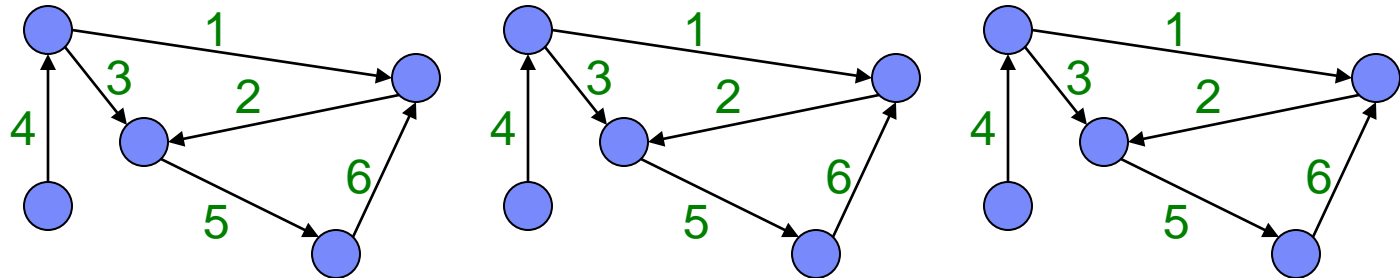
capacity constraints identify arcs and help to map nodes

# Multi-Commodity-Flow Cuts – Network Detection

## 2. Detect arcs

1	1	-C
1	1	-C

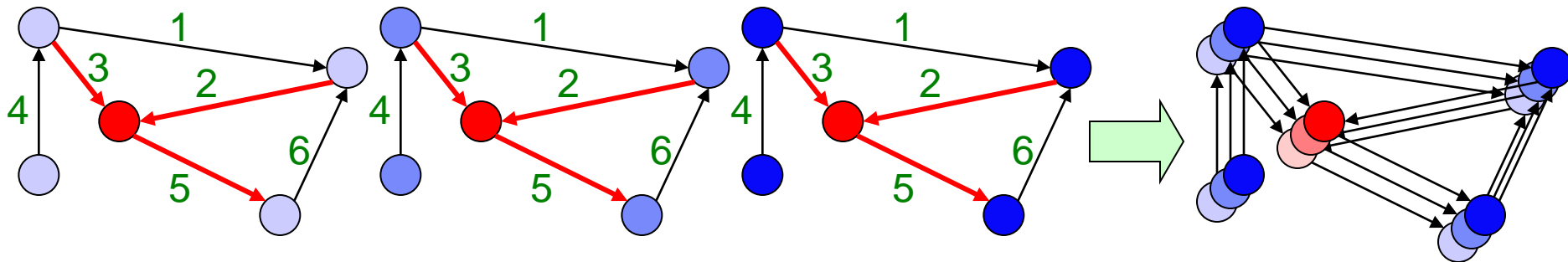
- identify capacity constraints and sort candidates by score
  - rows with (ideally) one +1 per commodity and binary or integer variables
- pick capacity constraints to define the arcs
- result:
  - directed graph, one component per commodity
  - mapping of capacity constraints to arcs



# Multi-Commodity-Flow Cuts – Network Detection

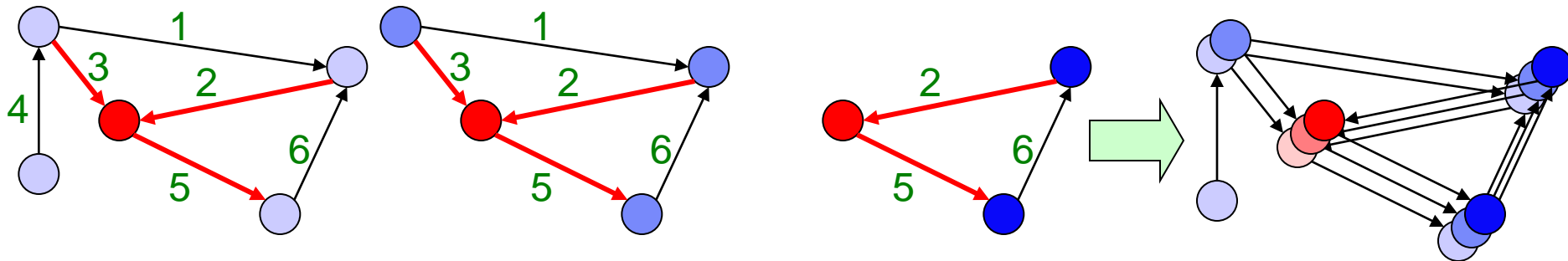
## 3. Detect nodes

- identify similar nodes in commodities with respect to the **arc incidence pattern**
- map each set of similar nodes to node in network graph
- result:
  - directed graph, one component per commodity
  - mapping of capacity constraints to arcs
  - mapping of flow constraints to nodes



# Multi-Commodity-Flow Cuts – Network Detection

- In reality, matrices are far from ideal network matrices
  - additional constraints and variables that do not belong to the network
  - user preprocessing
    - for example, omit one flow equation in each commodity
  - solver preprocessing
    - aggregate flow variables, discard rows and columns, ...

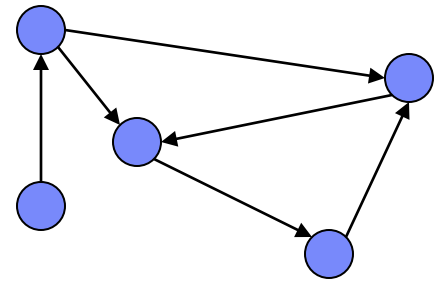


measure quality of network detection

⇒ do not separate cuts on poorly detected networks

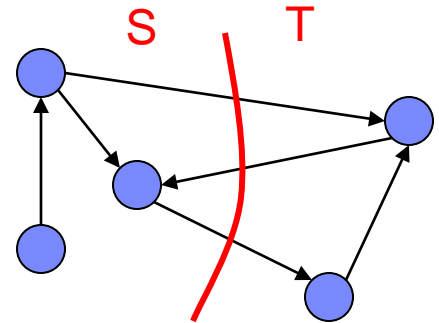
# Multi-Commodity-Flow Cuts – Cut Separation

- Apply procedures similar to network design literature
  - Bienstock and Günlük (1996)
  - Bienstock, Chopra, Günlük, Tsai (1998)
  - Günlük (1999)
  - Ortega and Wolsey (2003)
  - Raack, Koster, Orłowski, Wessäly (2007)
- Our heuristic network detection is error-prone
  - we cannot use the network directly
  - need to “emulate” network reasoning algebraically
    - aggregation of rows
    - apply cut procedure like c-MIR to aggregation

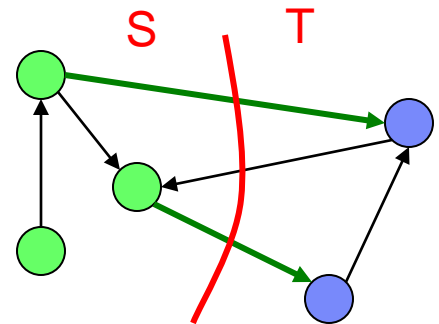


# Multi-Commodity-Flow Cuts – Cut Separation

- Consider cuts in the network graph
  - contract arcs until 5 nodes remain
  - consider all cuts in this cluster graph
    - cut idea: total capacity on  $S \rightarrow T$  arcs must satisfy demand in  $T$
  - additionally, generate flow cutset inequalities



- Construct cut algebraically
  - sum up flow conservation constraints in  $S$ 
    - flow inside of  $S$  cancels out
  - add capacity constraints for arcs in  $\delta^+(S)$ 
    - cancel flow variables, replace by capacity variables
  - apply c-MIR procedure
    - exploit integrality of capacity variables



# Multi-Commodity-Flow Cuts – Network Models

Test Set	#	faster	slower	time	nodes	#affect	time	nodes
arcset	30	15	4	0.74	0.45	23	0.68	0.36
cutset	15	0	0	1.00	1.00	0	1.00	1.00
fc	30	15	7	1.04	0.97	20	1.04	0.97
fctp	32	3	5	1.02	0.94	15	1.05	0.91
nexp	60	18	2	0.62	0.57	40	0.49	0.40
sndlib	52	19	1	0.68	0.66	22	0.40	0.32
ufcn	58	3	1	1.04	1.12	59	1.05	1.18

61% speed-up

47% speed-up

# Multi-Commodity-Flow Cuts – General MIPs

Test Set	#	faster	slower	time	nodes	#affect	time	nodes
MIPLIB 3 / 2003	92	3	1	0.99	0.98	8	0.93	0.77
Mittelmann	59	3	2	0.97	0.96	6	0.72	0.67
[0,1)	1018	3	2	1.00	0.99	36	0.99	0.82
[1,10)	486	15	10	≈50% speed-up		38	0.98	0.84
[10,100)	369	10	12	0.99	0.98	35	0.86	0.81
[100,1k)	284	12	7	0.97	0.96	24	0.66	0.60
[1k,10k)	192	13	7	0.96	0.96	22	0.68	0.71

13 unaffected with more than 2% degradation

7% affected models

# New Ingredients in CPLEX 12

- Connectors to Excel, MATLAB, Python
- Pseudocost updates for infeasible nodes
  - work of Emilie Danna and Andrea Lodi
- Multi-commodity-flow cuts
  - joint work with Christian Raack (ZIB)
  - also available in SCIP 1.1
    - check out CPLEX source code: [scip/src/scip/sepa\\_mcf.c](#)
    - about 6000 lines of C code
- A lot of additional improvements
  - but: performance results that show their effect

# Comparison of CPLEX 11 and 12 – MIP sequential

sup(time)	#	time limit hits	faster	slower	time	nodes	
[0,1)	985	—	67	33	0.99	1.18	
[1,10)	469	—	203	122	0.92	1.12	
[10,100)	391	—	198	121	0.82	0.94	
[100,1k)	284	—	32% speed-up		0.77	0.78	
[1k,10k)	201	—	115	57	0.59	0.67	
[1,10k}	1436	29	(-33)	715	427	0.76	0.86
[100,10k}	576	29	(-22)	314	194	0.62	0.65
[1k,10k}	292	29	(-33)	177	92	0.50	0.54

# Comparison of CPLEX 11 and 12 – MIP det. 4 threads

sup(time)	#	time limit hits	faster	slower	time	nodes	
[0,1)	1016	—	67	30	0.99	1.15	
[1,10)	505	—	227	124	0.91	0.98	
[10,100)	397	—	208	118	0.79	0.79	
[100,1k)	272	—	30% speed-up		0.74	0.63	
[1k,10k)	188	—	103	58	0.64	0.65	
[1,10k}	1475	35	(-43)	748	424	0.77	0.77
[100,10k}	573	35	(-42)	312	192	0.65	0.61
[1k,10k}	301	35	(-18)	178	92	0.57	0.58

Annotations: A red box labeled "30% speed-up" points to the transition between [100,1k) and [1k,10k). A red box labeled "54% speed-up" points to the transition between [100,10k) and [1k,10k). The values 0.77 and 0.65 in the "time" column are circled in red.

# CPLEX 12 – Speed-up Summary

algorithm	$\geq 1$ s	$\geq 10$ s	$\geq 100$ s	$\geq 1000$ s
LP Primal Simplex	no performance improvements			
LP Dual Simplex	3 %	4 %	4 %	9 %
LP Barrier (1 thread)	6 %	7 %	9 %	12 %
LP Barrier (4 threads)	22 %	23 %	18 %	10 %
QP (Barrier 1 thread)	6 %	8 %	11 %	20 %
MIP (1 thread)	32 %	45 %	61 %	101 %
MIP (determ. 4 threads)	30 %	43 %	54 %	74 %
MIQP (1 thread)	21 %	24 %	48 %	138 %
MIQCP (1 thread)	6 %	1 %	14 %	23 %