Mathematical Tools
for Engineering and Management

Lecture 7

30 Nov 2011

- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
- ▷ Sensitivity Analysis; (Mixed) Integer Programming
- ▷ MIP Modelling
- ▷ MIP Modelling: More Examples; Branch & Bound
- ▷ Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- ▷ Complexity Theory
- ▷ Nonlinear Optimization
- ▷ Scheduling
- ▷ Lot Sizing
- ▷ Multicriteria Optimization
- ▷ Oral exam

- Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

- Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

	modelling flexibility	solvability	optimality certificate
LPs	–	++	+++
MIPs	+	o	++
heuristics	++	+	– –
approximation algorithms	++	+	+
non-linear (convex) models	+	o	+
non-linear (general) models	++	–	o

▷ Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

	modelling flexibility	solvability	optimality certificate
LPs	–	++	+++
MIPs	+	o	++
heuristics	++	+	– –
approximation algorithms	++	+	+
non-linear (convex) models	+	o	+
non-linear (general) models	++	–	o

▷ Tuning branch & bound:

▷ Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

	modelling flexibility	solvability	optimality certificate
LPs	–	++	+++
MIPs	+	o	++
heuristics	++	+	– –
approximation algorithms	++	+	+
non-linear (convex) models	+	o	+
non-linear (general) models	++	–	o

▷ Tuning branch & bound:

- Order of processing the nodes (sub-problems)

▷ Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

	modelling flexibility	solvability	optimality certificate
LPs	–	++	+++
MIPs	+	o	++
heuristics	++	+	– –
approximation algorithms	++	+	+
non-linear (convex) models	+	o	+
non-linear (general) models	++	–	o

▷ Tuning branch & bound:

- Order of processing the nodes (sub-problems)
- Defining sub-problems (branching)

▷ Even if an optimal solution is not found, branch and bound delivers an optimality certificate, that gives valuable information on the solution quality!

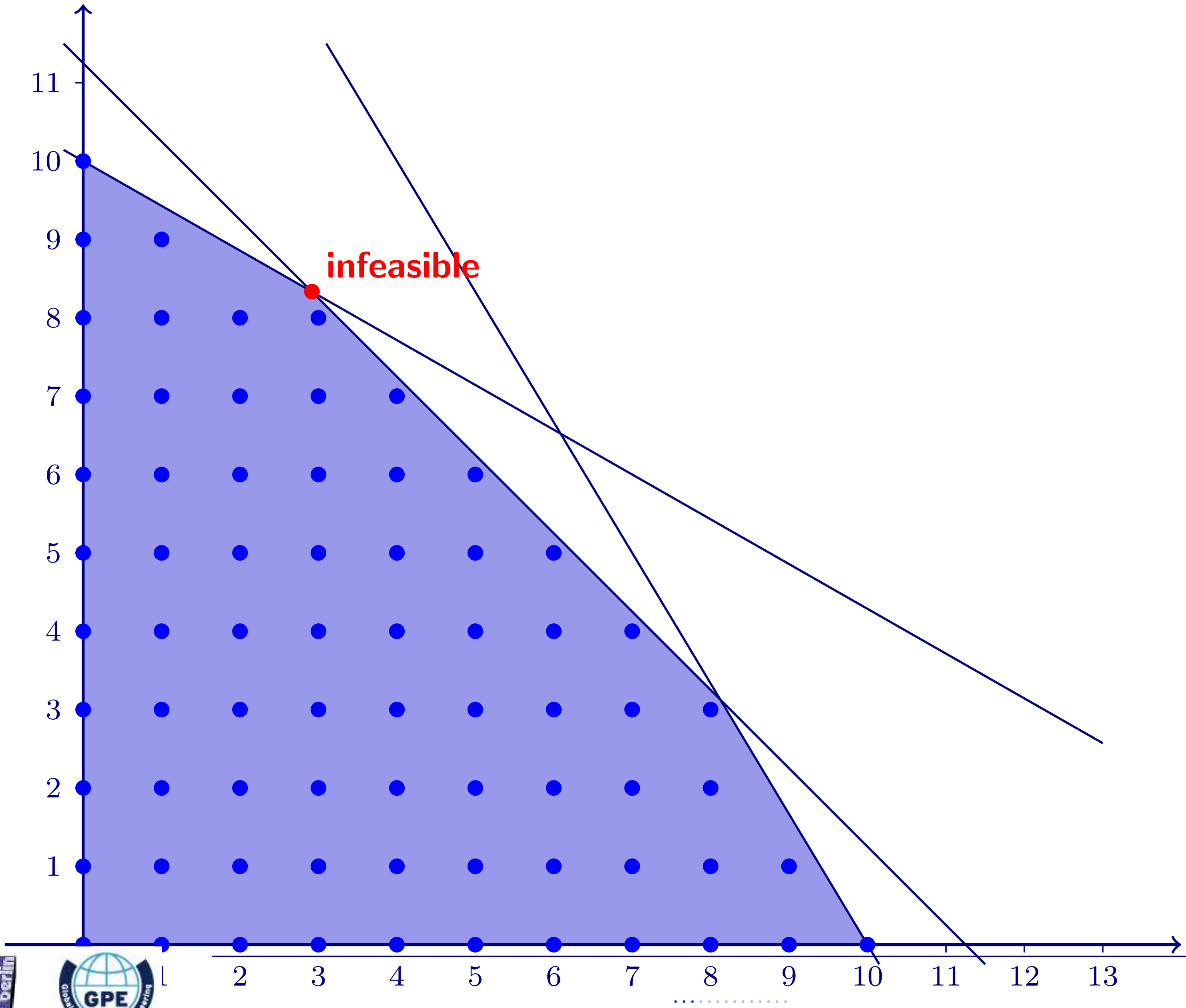
	modelling flexibility	solvability	optimality certificate
LPs	–	++	+++
MIPs	+	o	++
heuristics	++	+	– –
approximation algorithms	++	+	+
non-linear (convex) models	+	o	+
non-linear (general) models	++	–	o

▷ Tuning branch & bound:

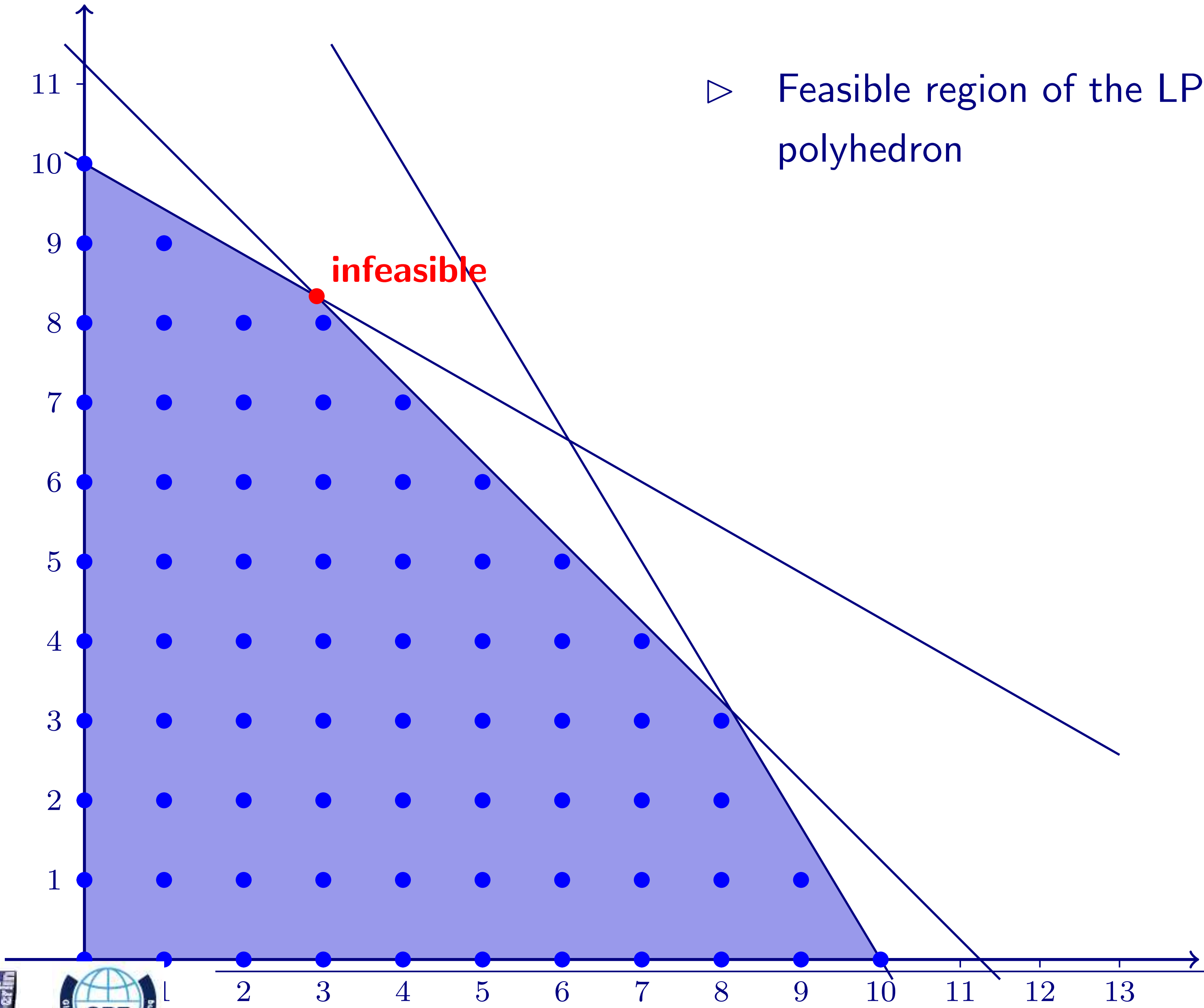
- Order of processing the nodes (sub-problems)
- Defining sub-problems (branching)
- Cutting planes ➡ branch & cut

=== cutting planes ===

=== cutting planes ===

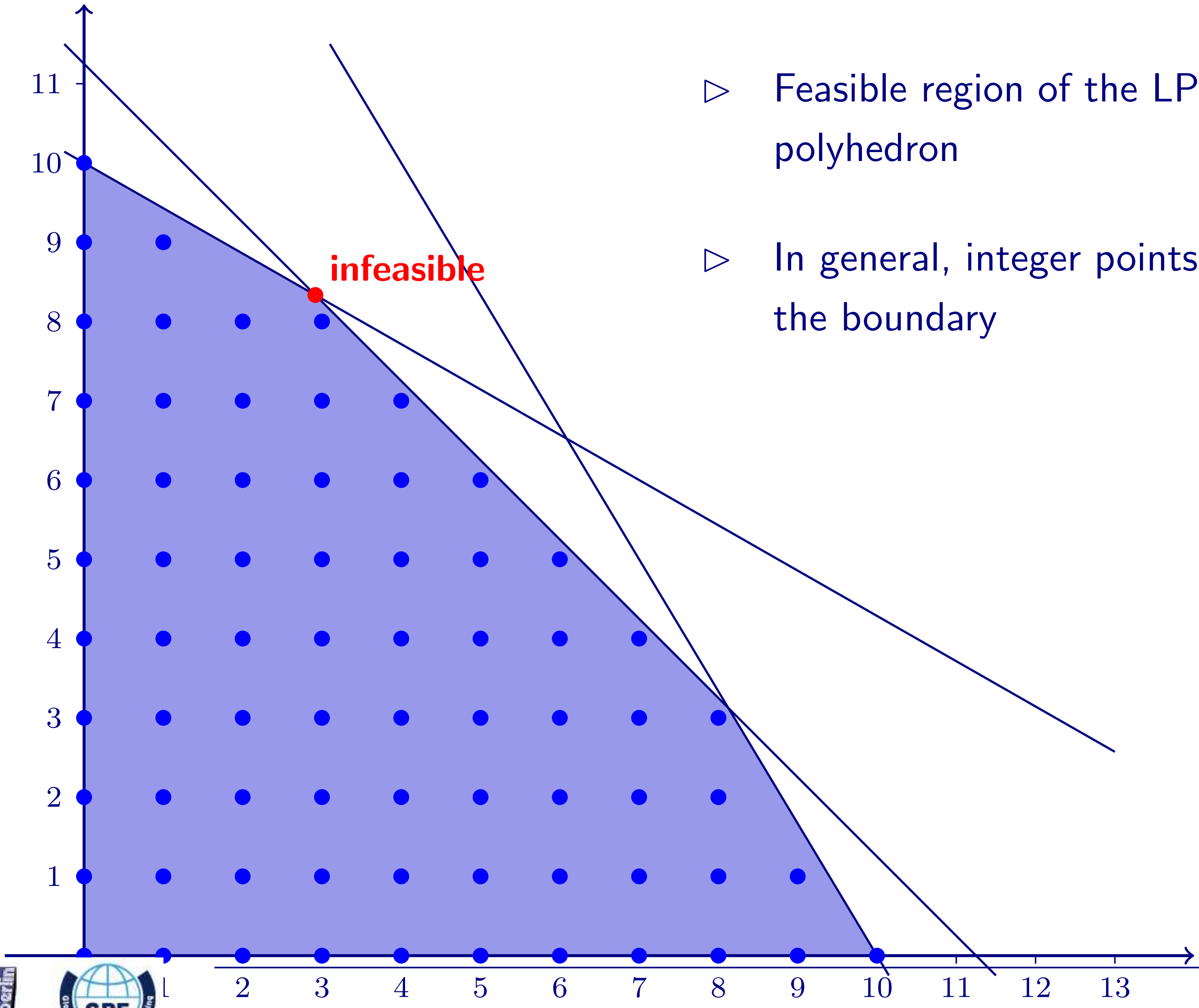


=== cutting planes ===

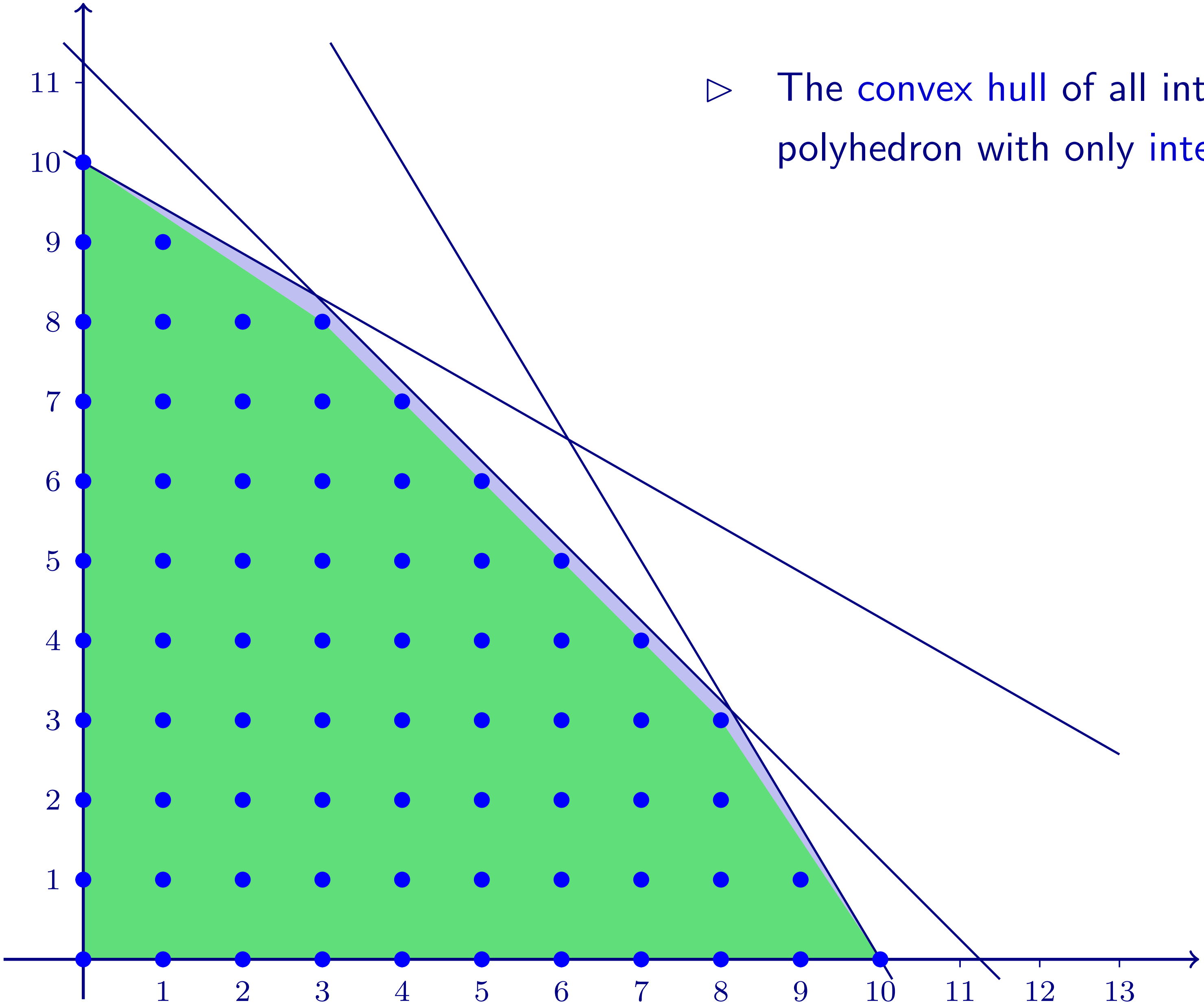


- ▷ Feasible region of the LP relaxation is a polyhedron

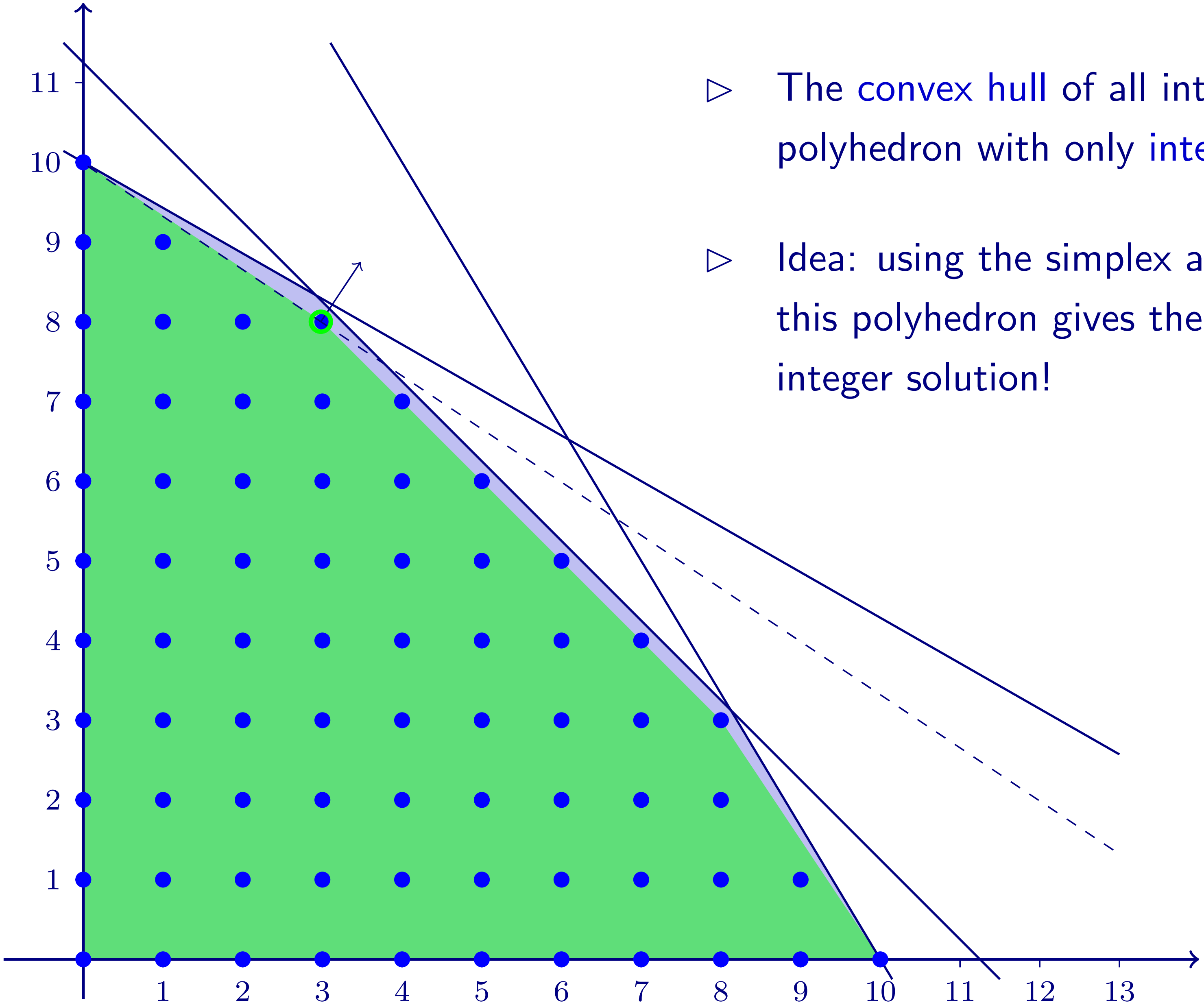
=== cutting planes ===



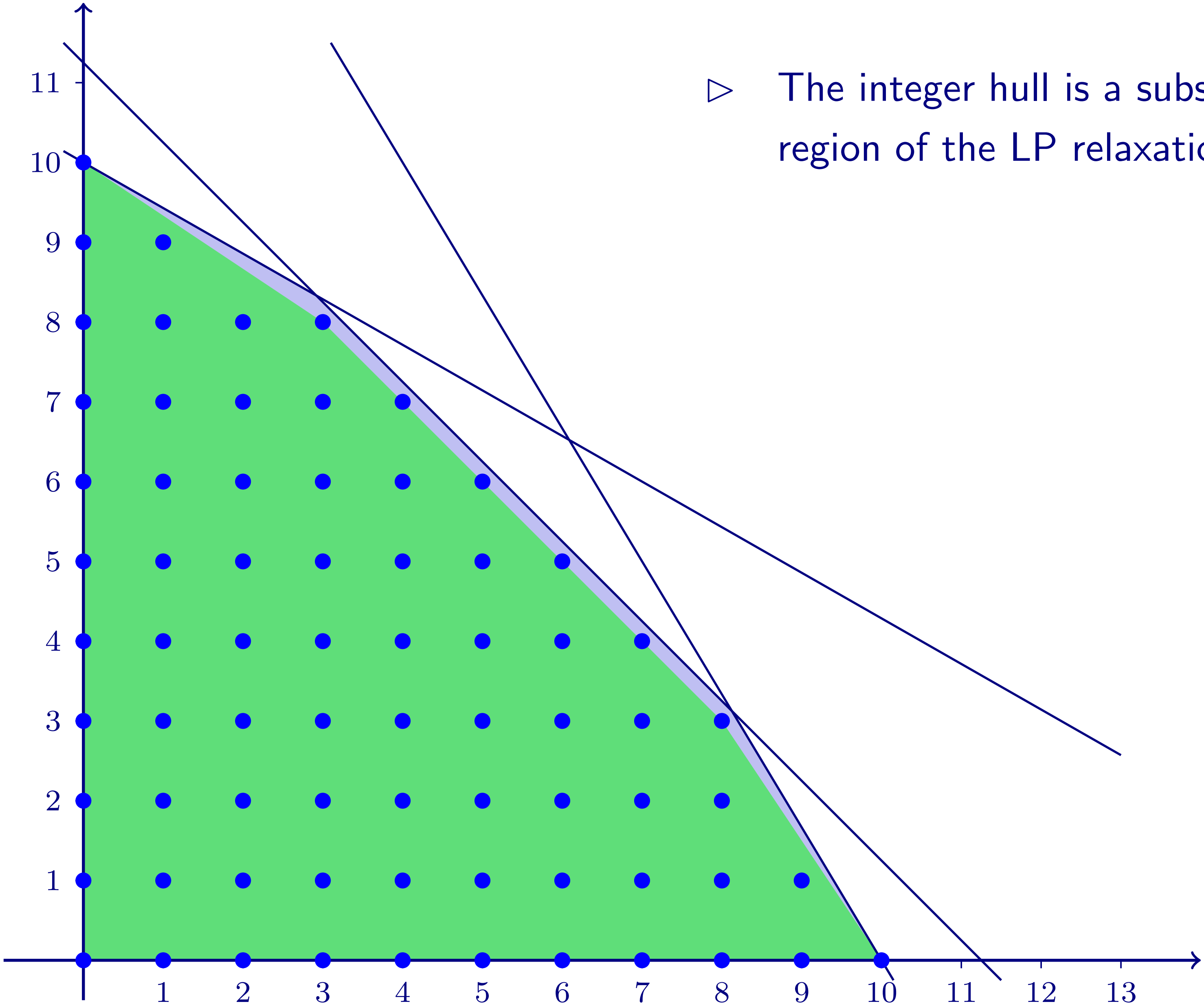
- Feasible region of the LP relaxation is a polyhedron
- In general, integer points do not lie on the boundary



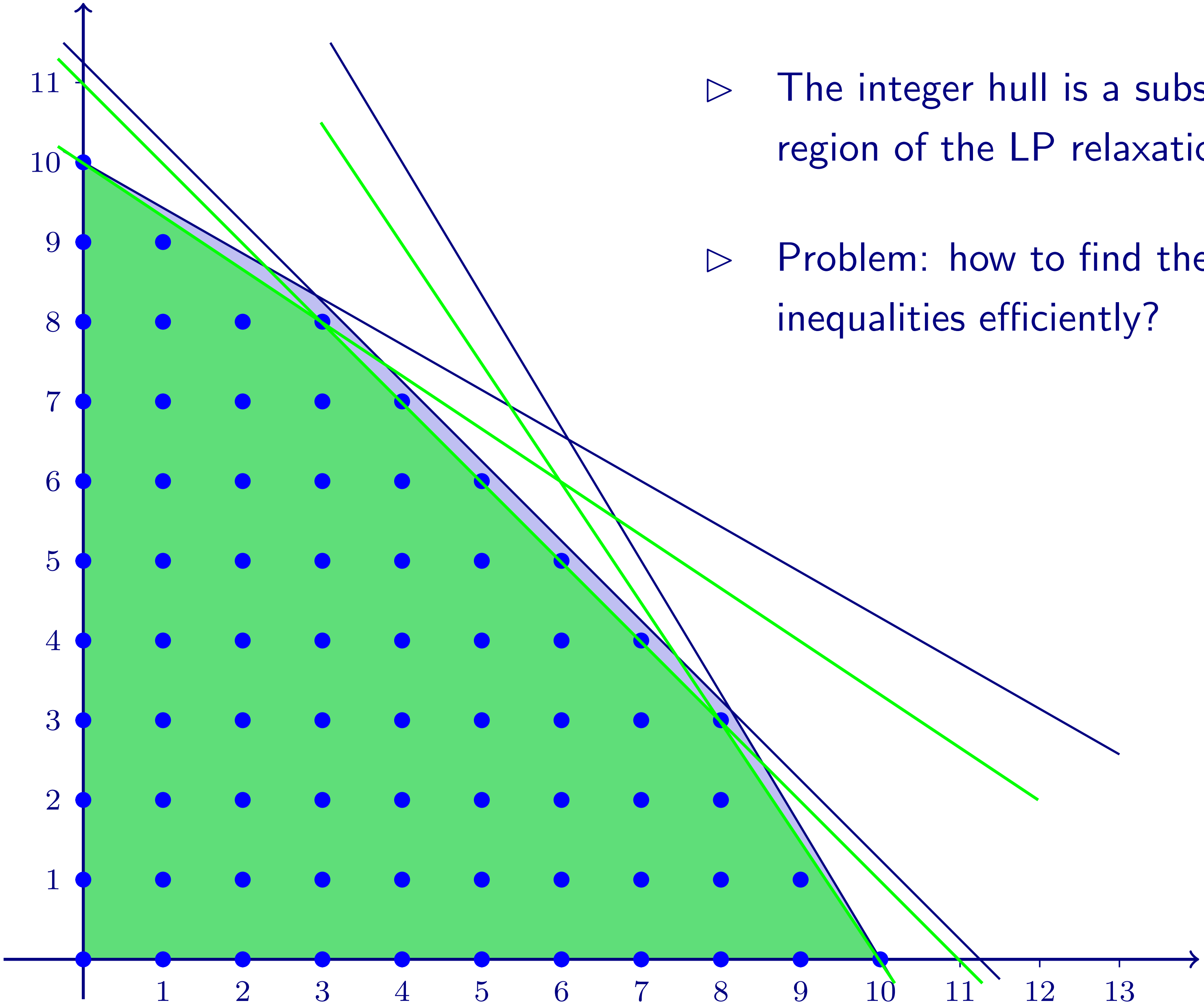
- The convex hull of all integer solutions is a polyhedron with only integer vertices



- The convex hull of all integer solutions is a polyhedron with only integer vertices
- Idea: using the simplex algorithm on this polyhedron gives the optimal integer solution!



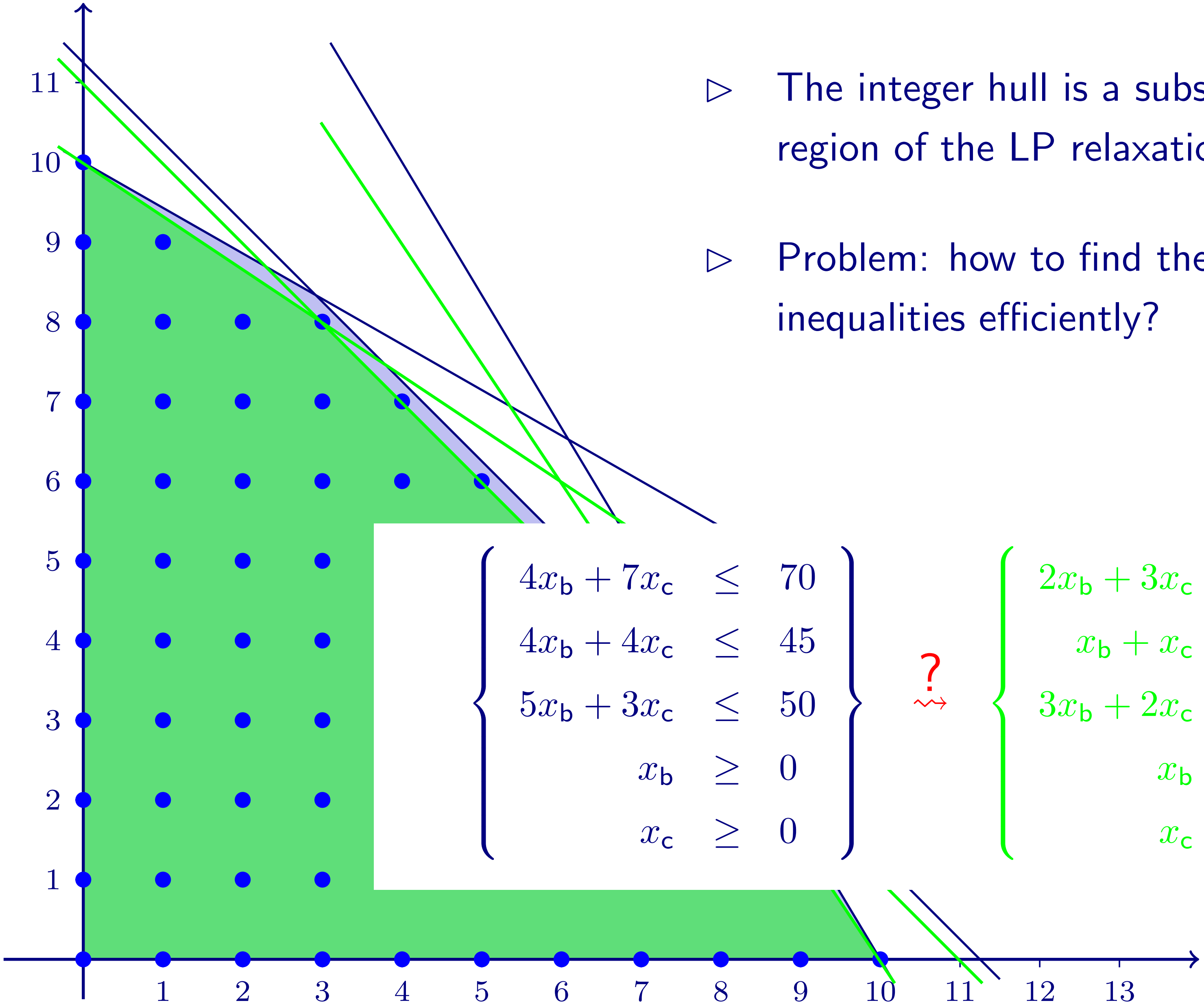
- The integer hull is a subset of the feasible region of the LP relaxation



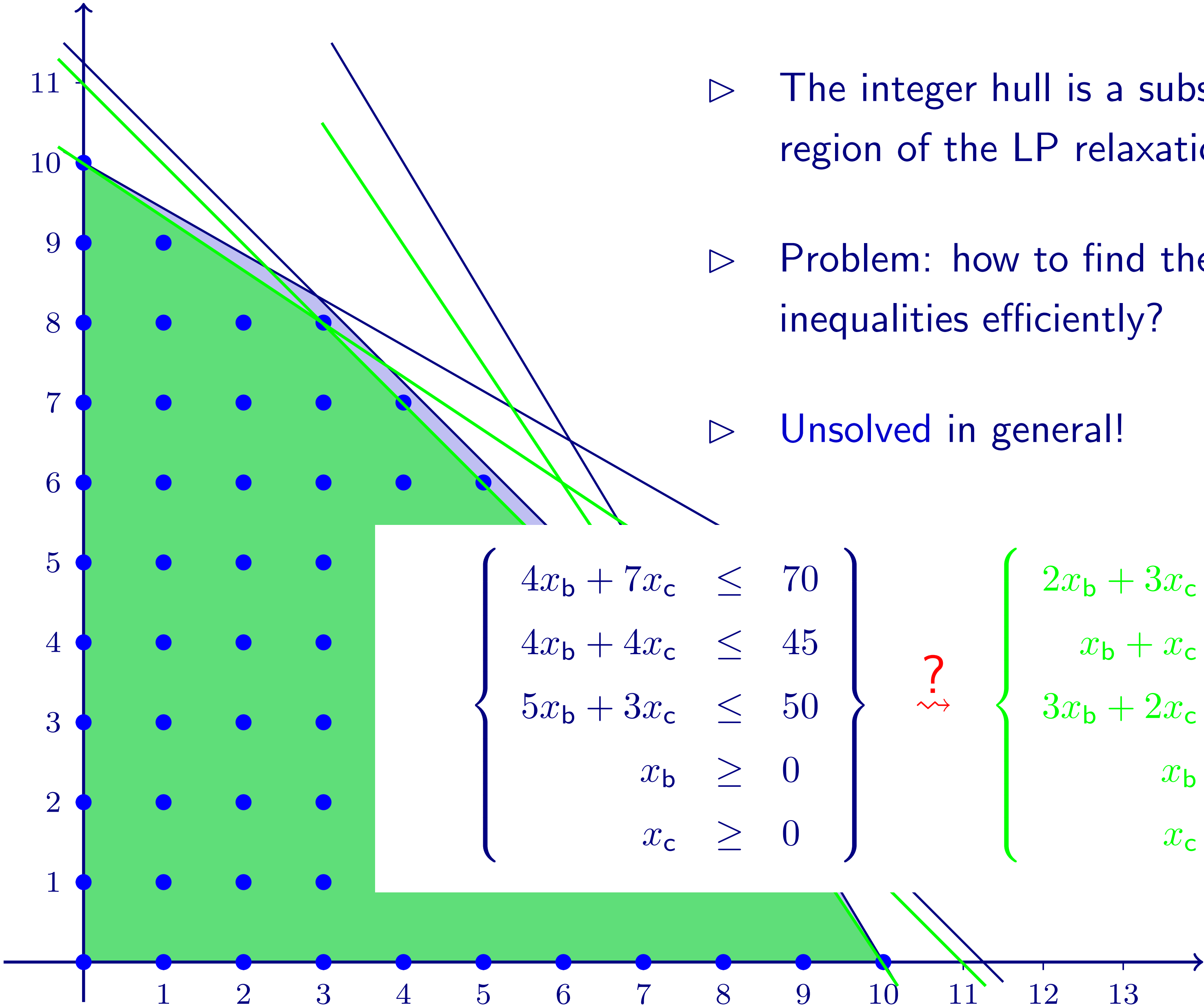
- ▷ The integer hull is a subset of the feasible region of the LP relaxation
- ▷ Problem: how to find the describing inequalities efficiently?

- The integer hull is a subset of the feasible region of the LP relaxation

- Problem: how to find the describing inequalities efficiently?

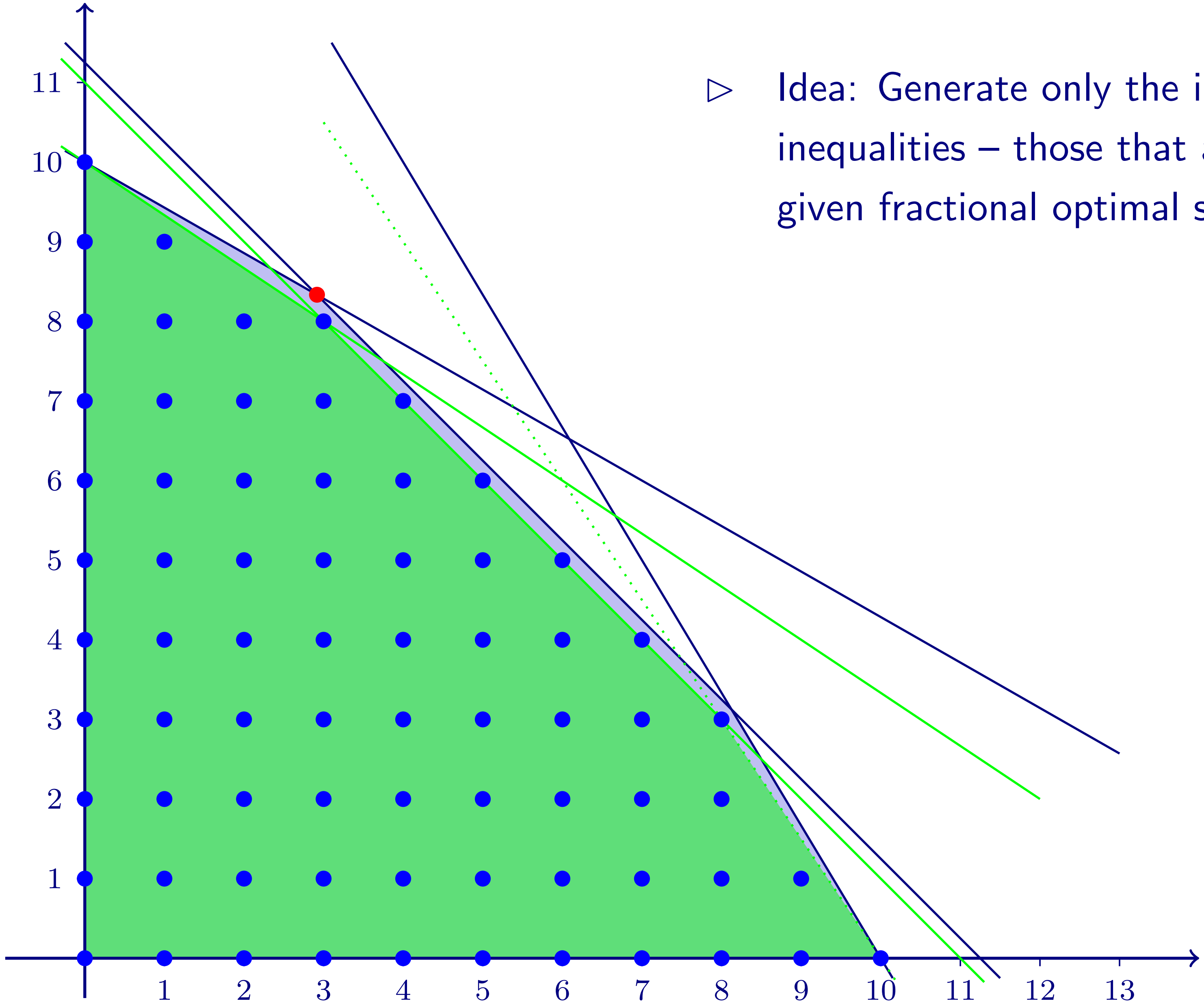


$$\left\{\begin{array}{rcl} 4x_{\mathsf{b}} + 7x_{\mathsf{c}} & \leq & 70 \\ 4x_{\mathsf{b}} + 4x_{\mathsf{c}} & \leq & 45 \\ 5x_{\mathsf{b}} + 3x_{\mathsf{c}} & \leq & 50 \\ x_{\mathsf{b}} & \geq & 0 \\ x_{\mathsf{c}} & \geq & 0 \end{array}\right\} \overset{?}{\rightsquigarrow} \left\{\begin{array}{rcl} 2x_{\mathsf{b}} + 3x_{\mathsf{c}} & \leq & 30 \\ x_{\mathsf{b}} + x_{\mathsf{c}} & \leq & 11 \\ 3x_{\mathsf{b}} + 2x_{\mathsf{c}} & \leq & 30 \\ x_{\mathsf{b}} & \geq & 0 \\ x_{\mathsf{c}} & \geq & 0 \end{array}\right\}$$

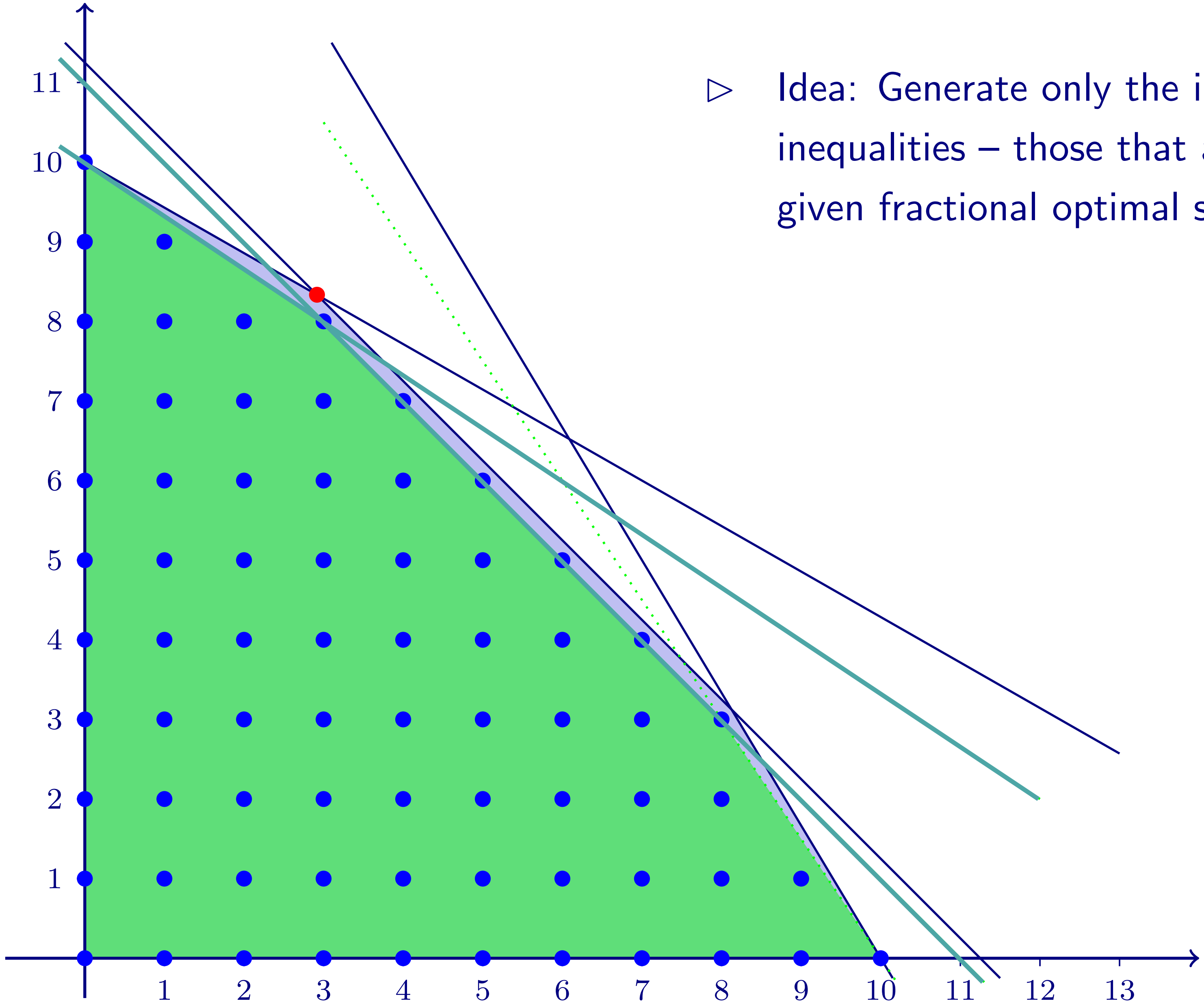


- The integer hull is a subset of the feasible region of the LP relaxation
- Problem: how to find the describing inequalities efficiently?
- Unsolved in general!

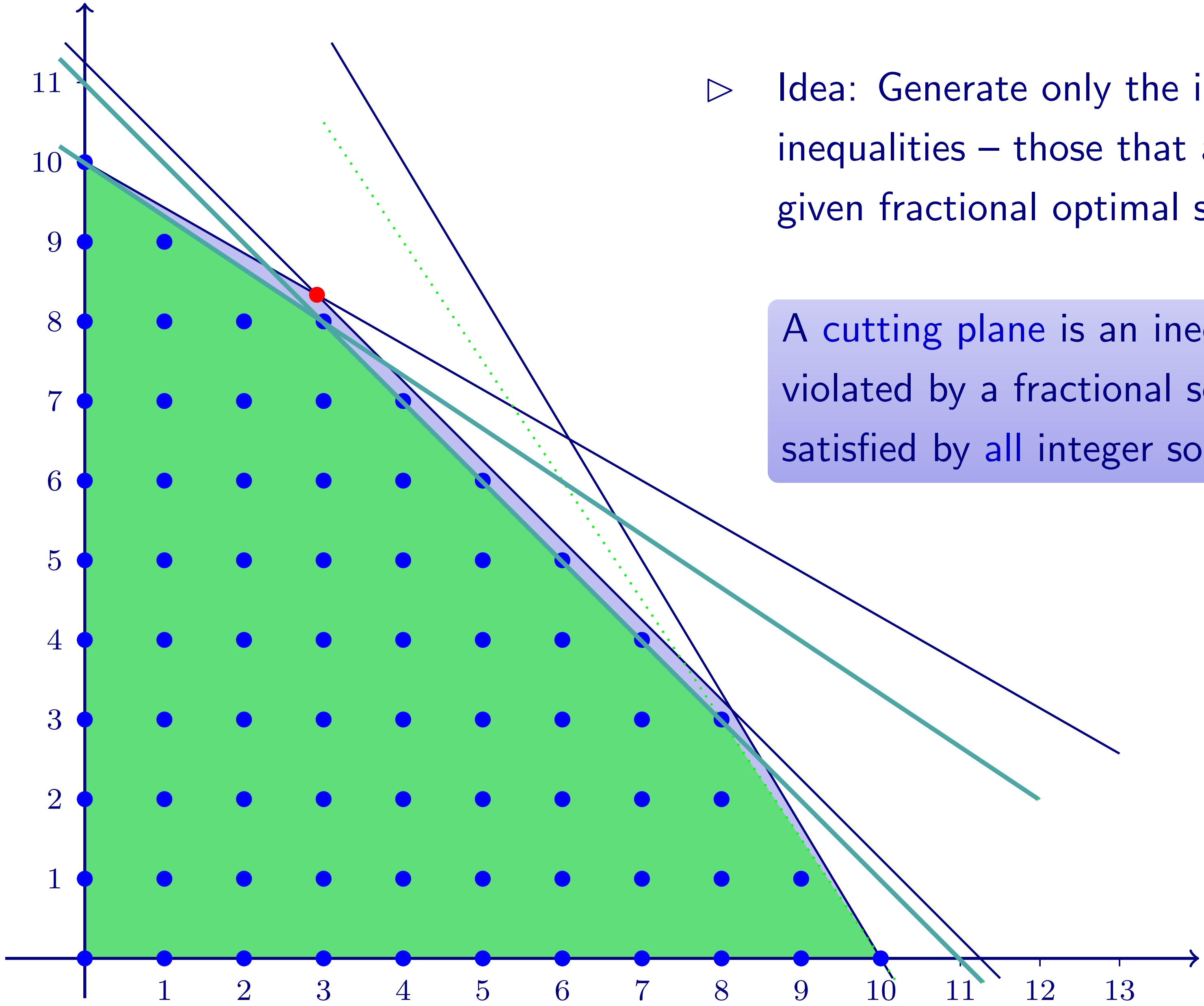
$$\left\{\begin{array}{rcl} 4x_\mathsf{b} + 7x_\mathsf{c} & \leq & 70 \\ 4x_\mathsf{b} + 4x_\mathsf{c} & \leq & 45 \\ 5x_\mathsf{b} + 3x_\mathsf{c} & \leq & 50 \\ x_\mathsf{b} & \geq & 0 \\ x_\mathsf{c} & \geq & 0 \end{array}\right\} \overset{?}{\rightsquigarrow} \left\{\begin{array}{rcl} 2x_\mathsf{b} + 3x_\mathsf{c} & \leq & 30 \\ x_\mathsf{b} + x_\mathsf{c} & \leq & 11 \\ 3x_\mathsf{b} + 2x_\mathsf{c} & \leq & 30 \\ x_\mathsf{b} & \geq & 0 \\ x_\mathsf{c} & \geq & 0 \end{array}\right\}$$



- Idea: Generate only the important inequalities – those that are violated by a given fractional optimal solution

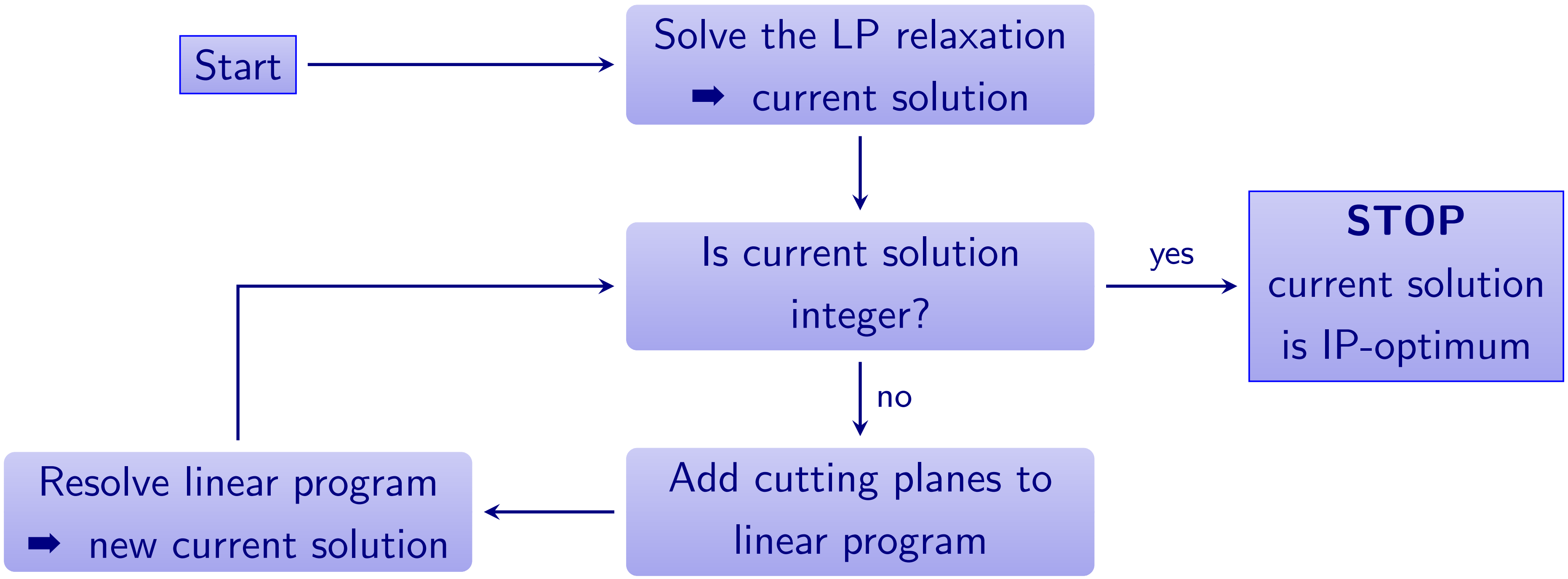


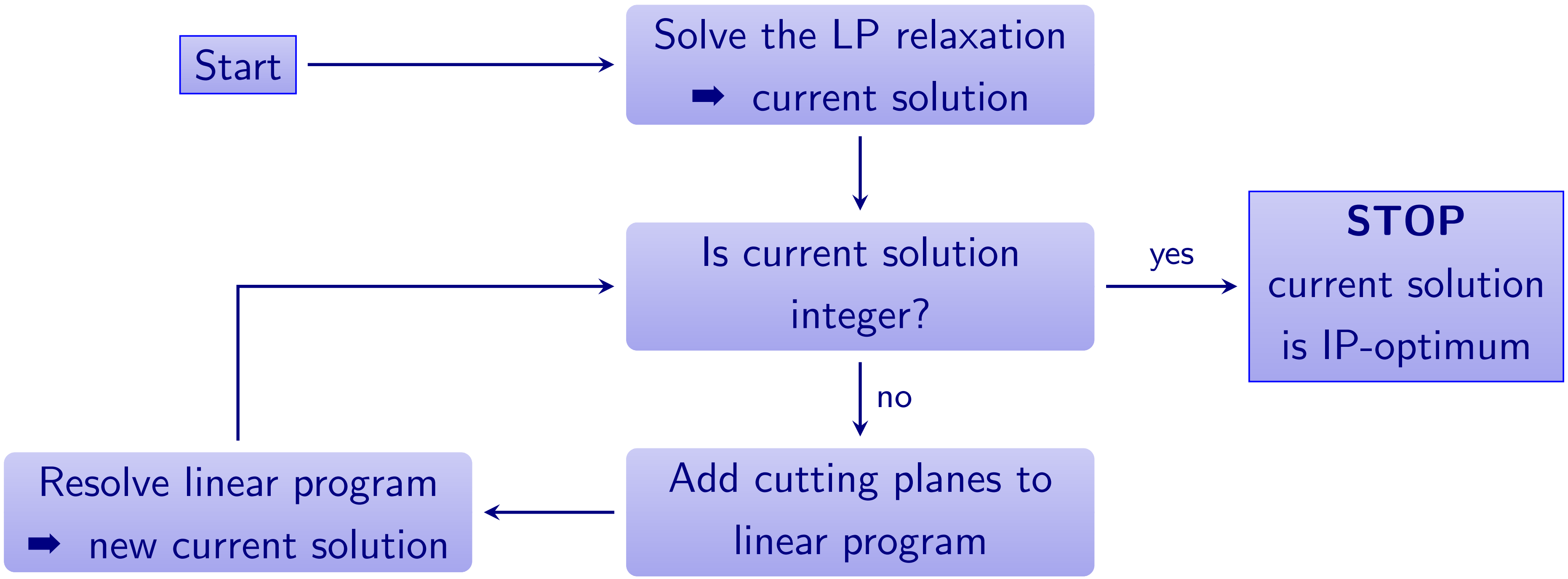
- ▷ Idea: Generate only the important inequalities – those that are violated by a given fractional optimal solution



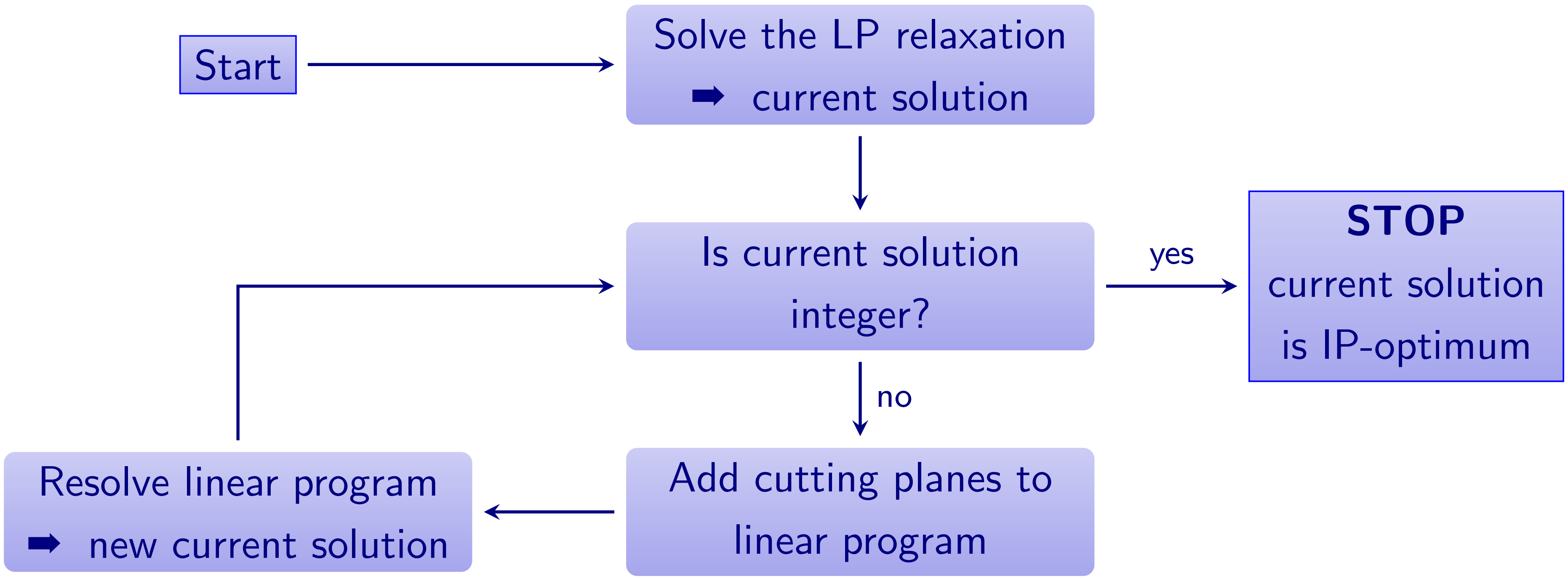
- Idea: Generate only the important inequalities – those that are violated by a given fractional optimal solution

A cutting plane is an inequality that is violated by a fractional solution but satisfied by all integer solutions

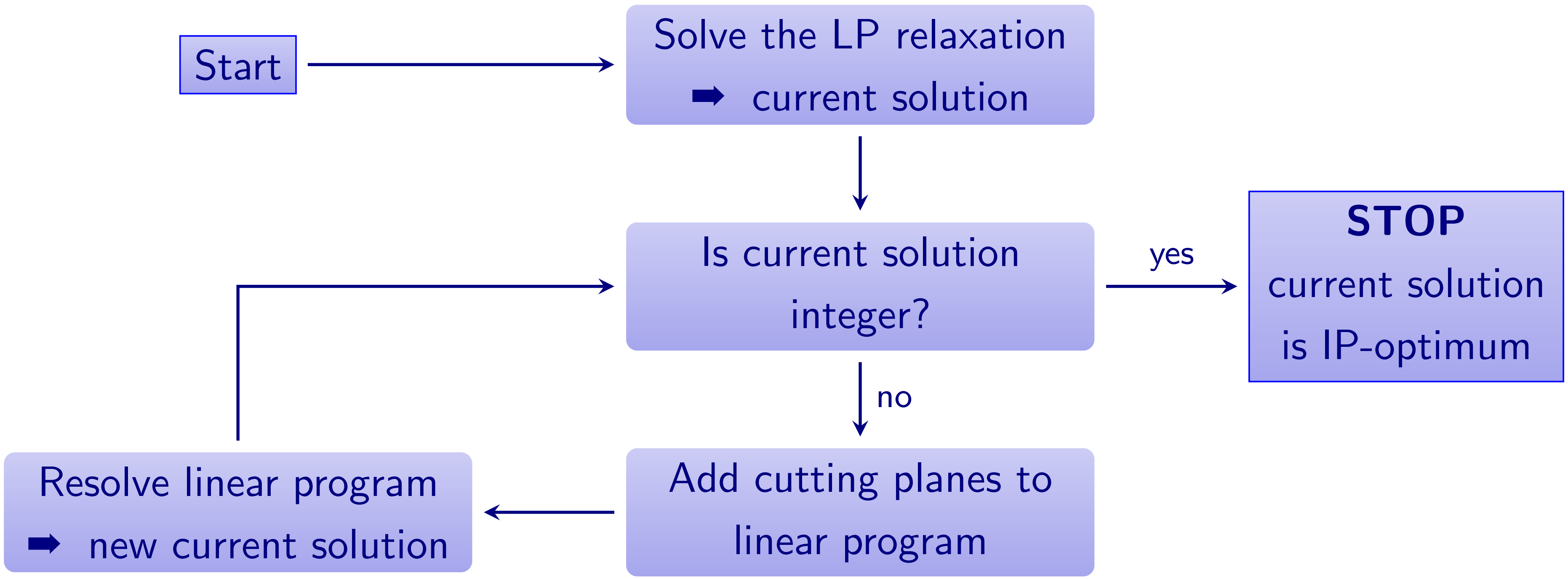

Start
Solve the LP relaxation
➡ current solution
Is current solution
integer?
yes
STOP
current solution
is IP-optimum
no
Add cutting planes to
linear program
Resolve linear program
➡ new current solution



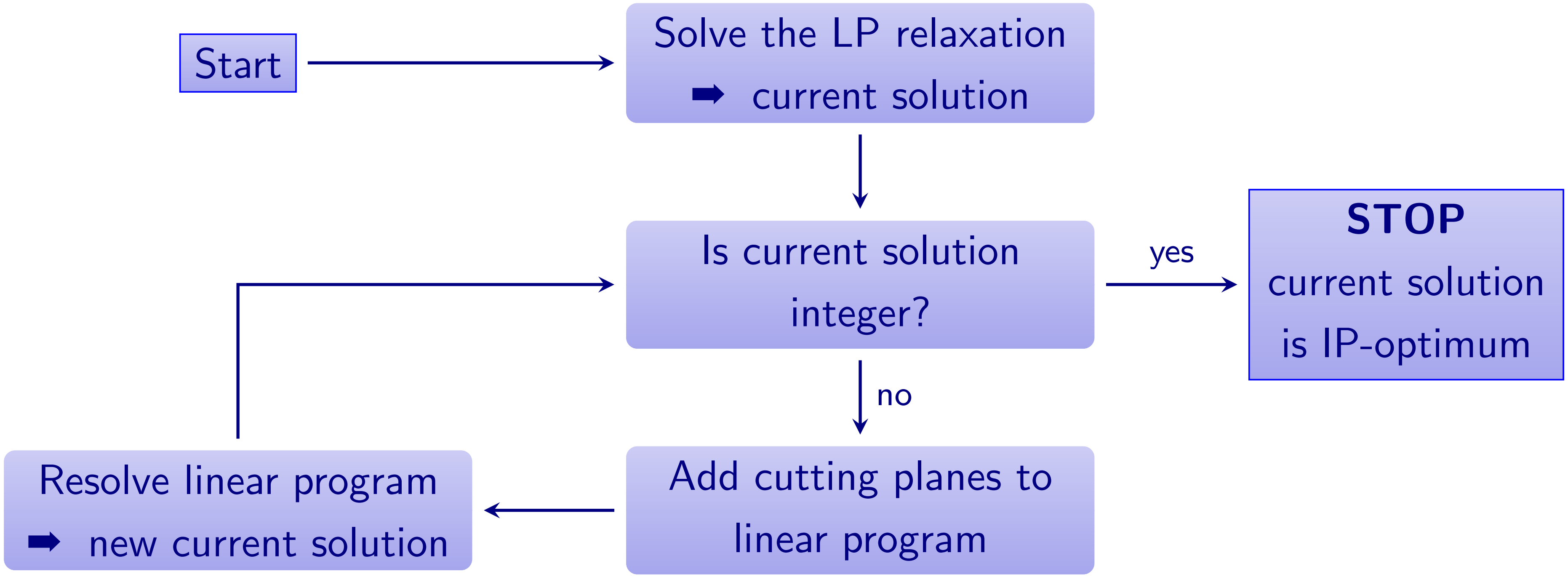
- Problem: no method known that effectively generates all cutting planes



- Problem: no method known that effectively generates all cutting planes
  - ➡ No guarantee that the algorithm terminates

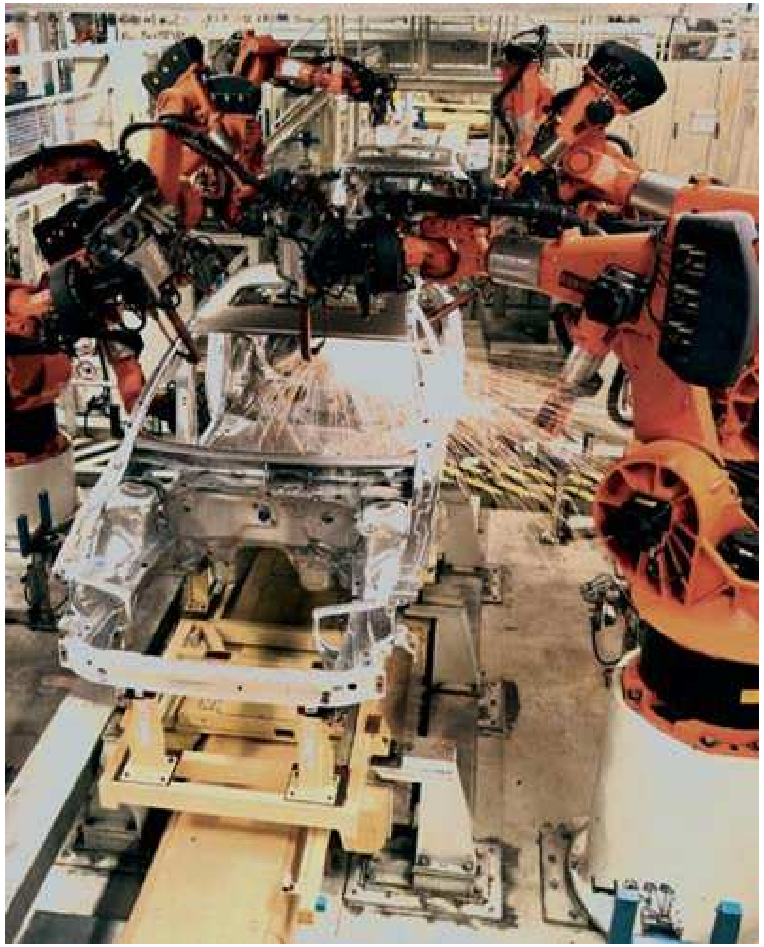


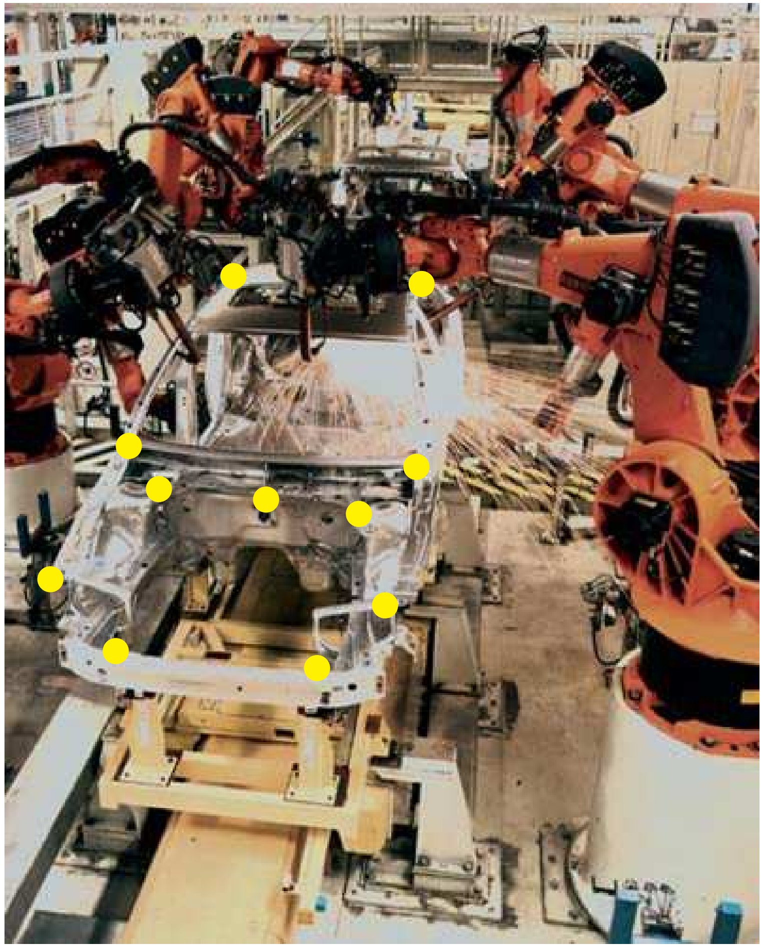
- Problem: no method known that effectively generates all cutting planes
  - ➡ No guarantee that the algorithm terminates
- Use cutting planes in combination with branch & bound

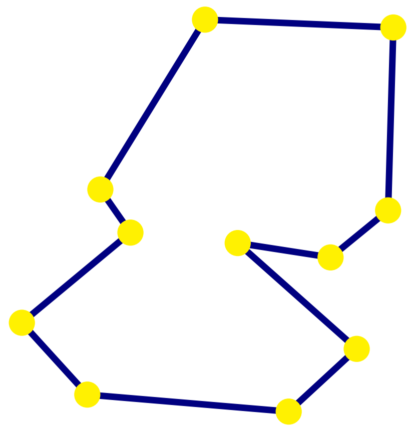


- ▷ Problem: no method known that effectively generates all cutting planes
  - ➡ No guarantee that the algorithm terminates
- ▷ Use cutting planes in combination with branch & bound
  - ➡ branch & cut: generate some “promising” cutting planes in tree nodes to improve subproblem solutions

Back to a previous example



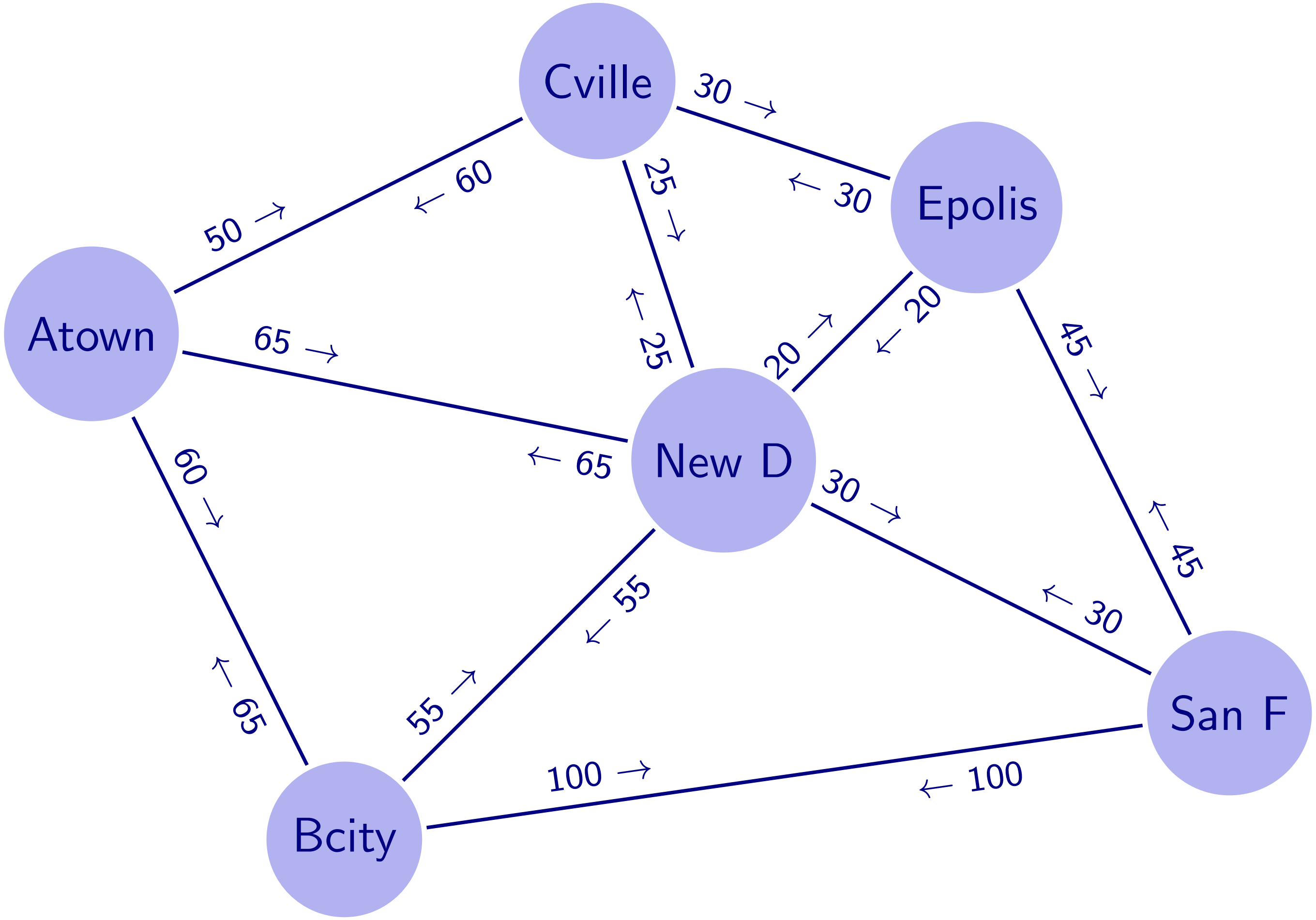




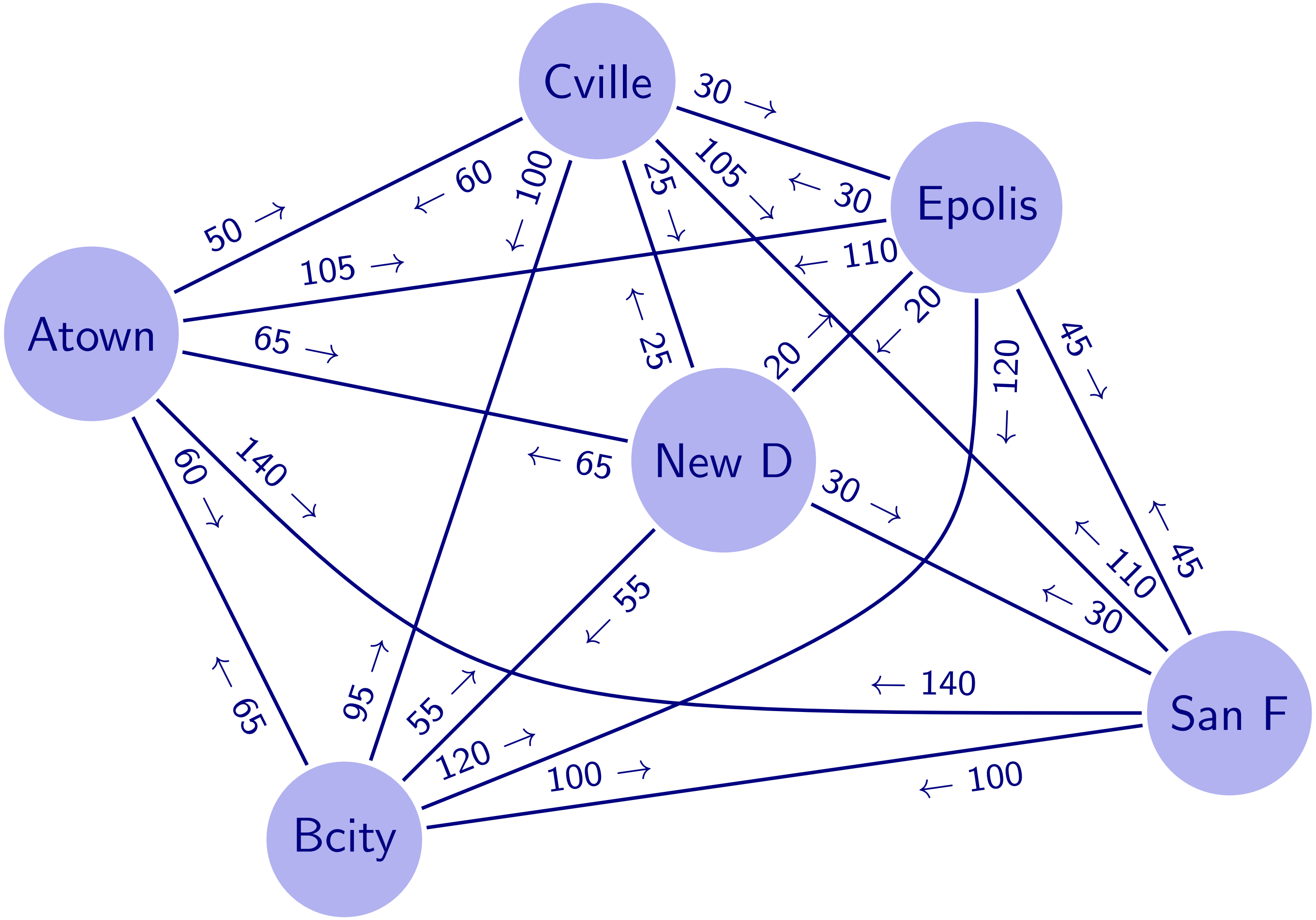
▷ Problem to solve: find an optimal order of welding points!

Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.

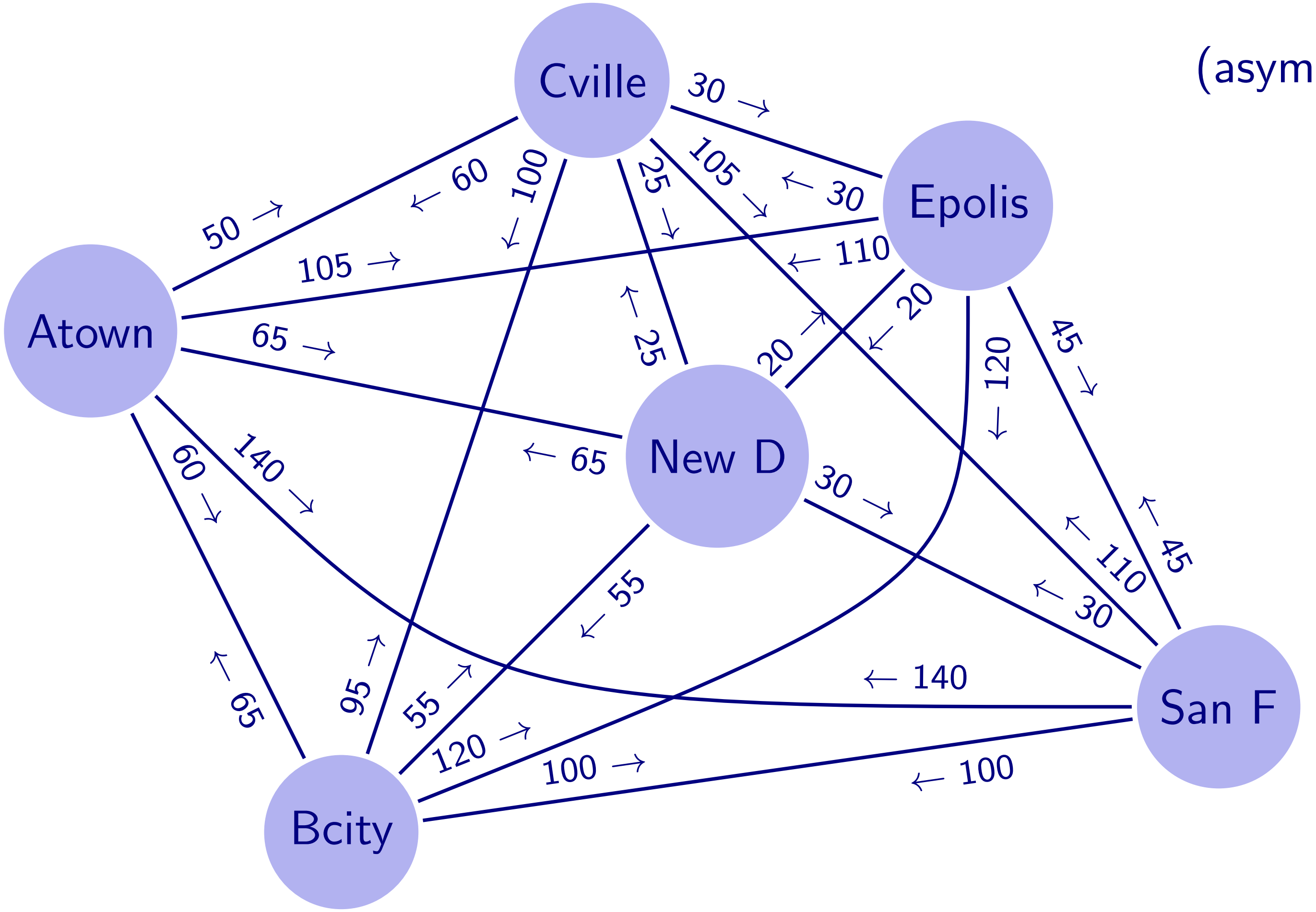
Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



(asymmetric TSP)

- A typical combinatorial problem...

- A typical combinatorial problem...
- Complete enumeration of all possible solutions is not an option (combinatorial explosion):

- A typical combinatorial problem...
- Complete enumeration of all possible solutions is not an option (combinatorial explosion):

# cities	# possible tours	time to try out all tours
5	24	0.012 milliseconds
10	362,880	0.18 seconds
15	87,178,291,200	12 hours
20	121,645,100,408,832,000	1927 years

- A typical combinatorial problem...
- Complete enumeration of all possible solutions is not an option (combinatorial explosion):

# cities	# possible tours	time to try out all tours
5	24	0.012 milliseconds
10	362,880	0.18 seconds
15	87,178,291,200	12 hours
20	121,645,100,408,832,000	1927 years

- TSP can be formulated as a binary integer program

- A typical combinatorial problem...
- Complete enumeration of all possible solutions is not an option (combinatorial explosion):

# cities	# possible tours	time to try out all tours
5	24	0.012 milliseconds
10	362,880	0.18 seconds
15	87,178,291,200	12 hours
20	121,645,100,408,832,000	1927 years

- TSP can be formulated as a binary integer program
  - ➡ Provides upper bounds for the solution

- ▷ A typical combinatorial problem...
- ▷ Complete enumeration of all possible solutions is not an option (combinatorial explosion):

# cities	**# possible tours**	**time to try out all tours**
5	24	0.012 milliseconds
10	362,880	0.18 seconds
15	87,178,291,200	12 hours
20	121,645,100,408,832,000	1927 years

- ▷ TSP can be formulated as a binary integer program
  - ➡ Provides upper bounds for the solution
  - ➡ Huge number of variables and constraints
  - ➡ Gives optimal solutions only for instances with small number of cities

- A typical combinatorial problem...
- Complete enumeration of all possible solutions is not an option (combinatorial explosion):

# cities	# possible tours	time to try out all tours
5	24	0.012 milliseconds
10	362,880	0.18 seconds
15	87,178,291,200	12 hours
20	121,645,100,408,832,000	1927 years

- TSP can be formulated as a binary integer program
  - ➡ Provides upper bounds for the solution
  - ➡ Huge number of variables and constraints
  - ➡ Gives optimal solutions only for instances with small number of cities
- Combinatorial algorithms are much more successful
  - ➡ Combinatorial optimization

▷ Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

- Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

  Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

- Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

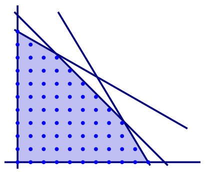
> Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

- Problems can often be solved with integer programming models

- Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

  Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

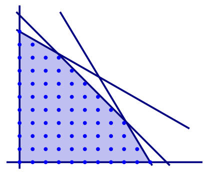
- Problems can often be solved with integer programming models
  ➡ Collection of objects = integer points in the feasible region



- Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

  > Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

- Problems can often be solved with integer programming models
  - ➡ Collection of objects = integer points in the feasible region



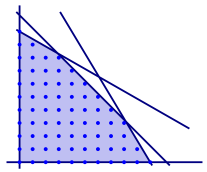
- But: usually there are special approaches that are much more efficient

- Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer 2003:

  Combinatorial optimization searches for an optimum object in a finite collection of objects. Typically, the collection has a concise representation (like a graph), while the number of objects is huge — more precisely, grows exponentially in the size of the representation (like all matchings or all Hamiltonian circuits). So scanning all objects one by one and selecting the best one is not an option. More efficient methods should be found.

- Problems can often be solved with integer programming models

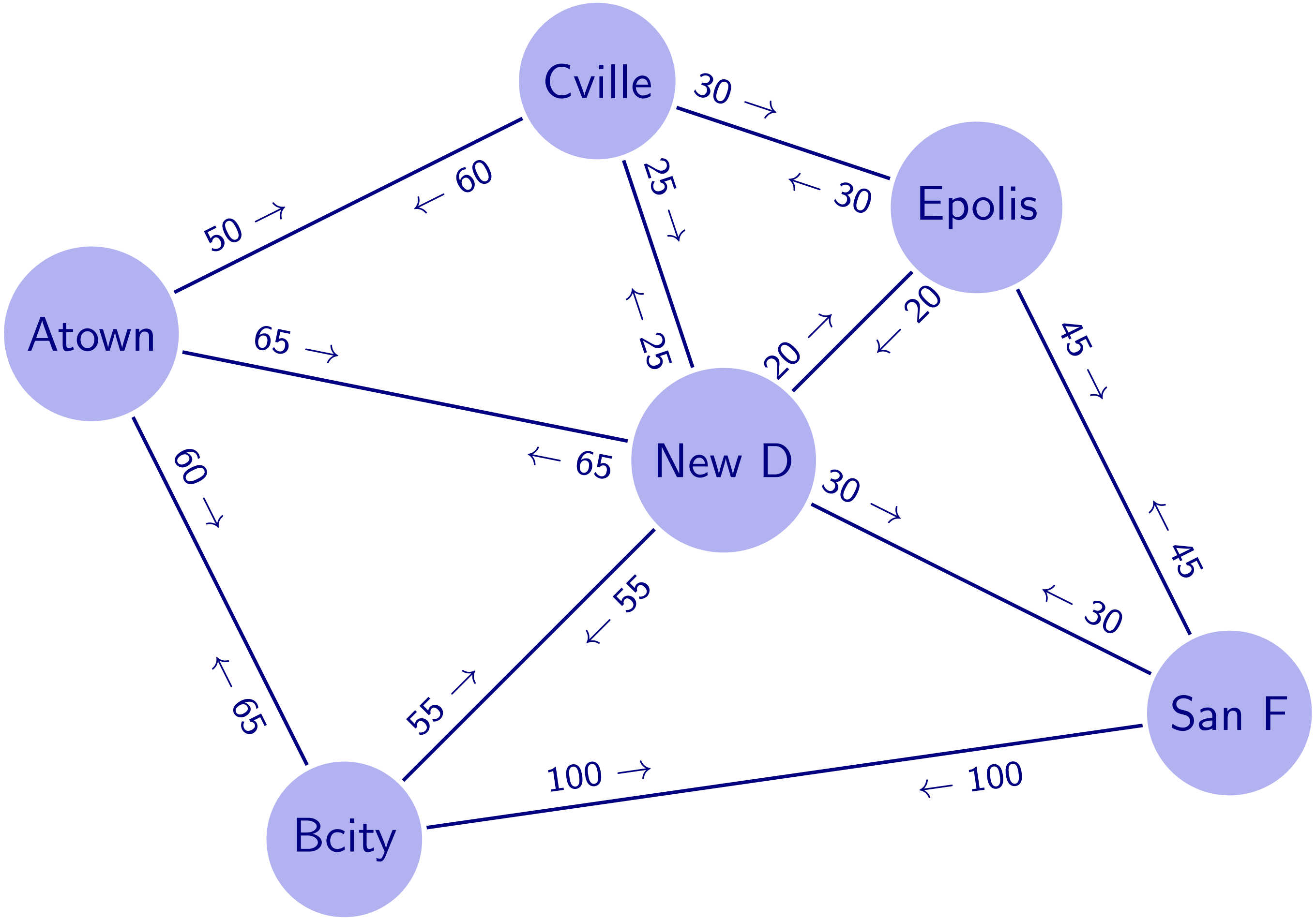
  ➡ Collection of objects = integer points in the feasible region



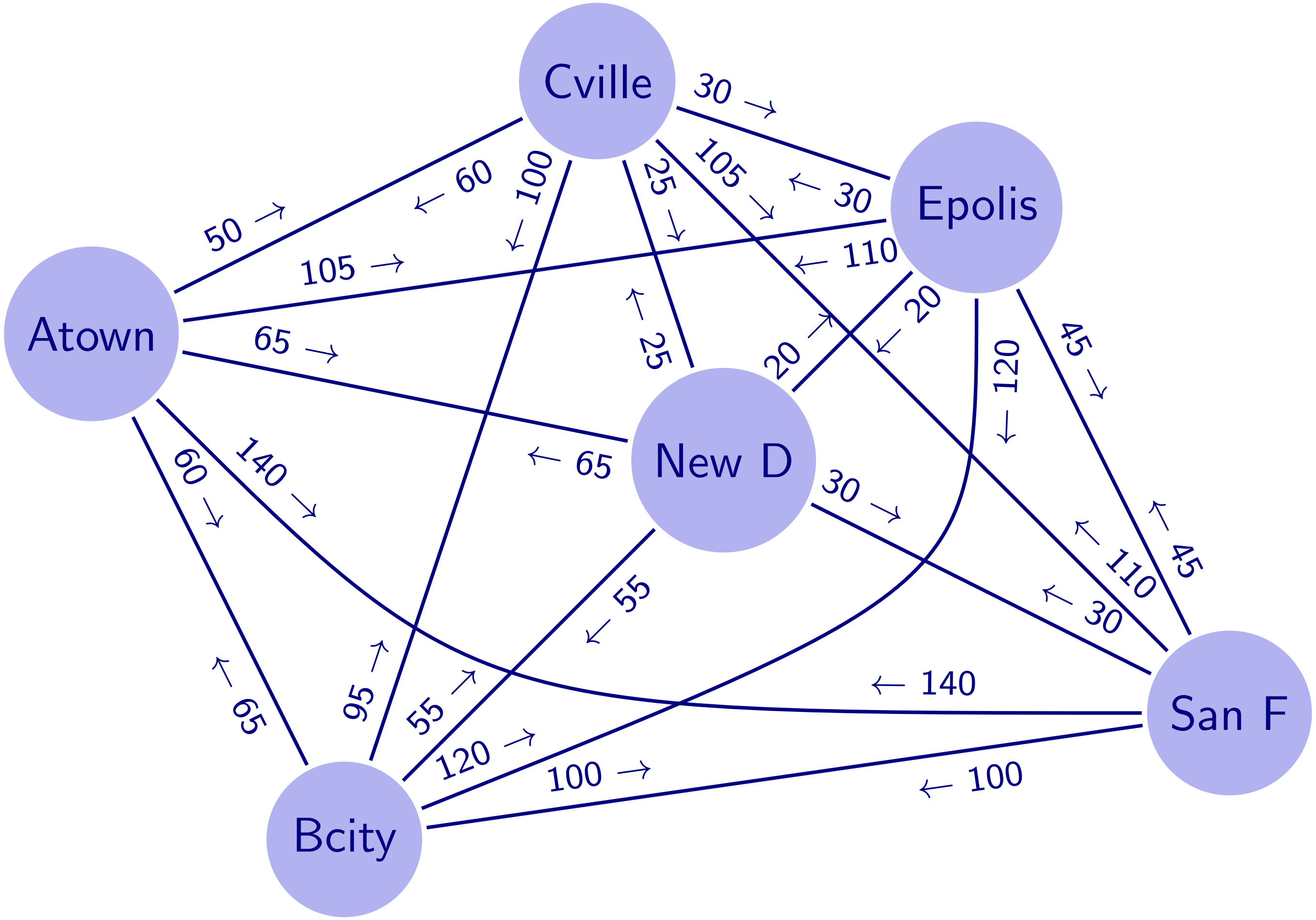
- But: usually there are special approaches that are much more efficient
- Examples: Travelling Salesman Problem, Minimum Spanning Tree, Knapsack Problem, Shortest Path Problem, Network Flow, Matching, Stable Set Problem, ...

Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.

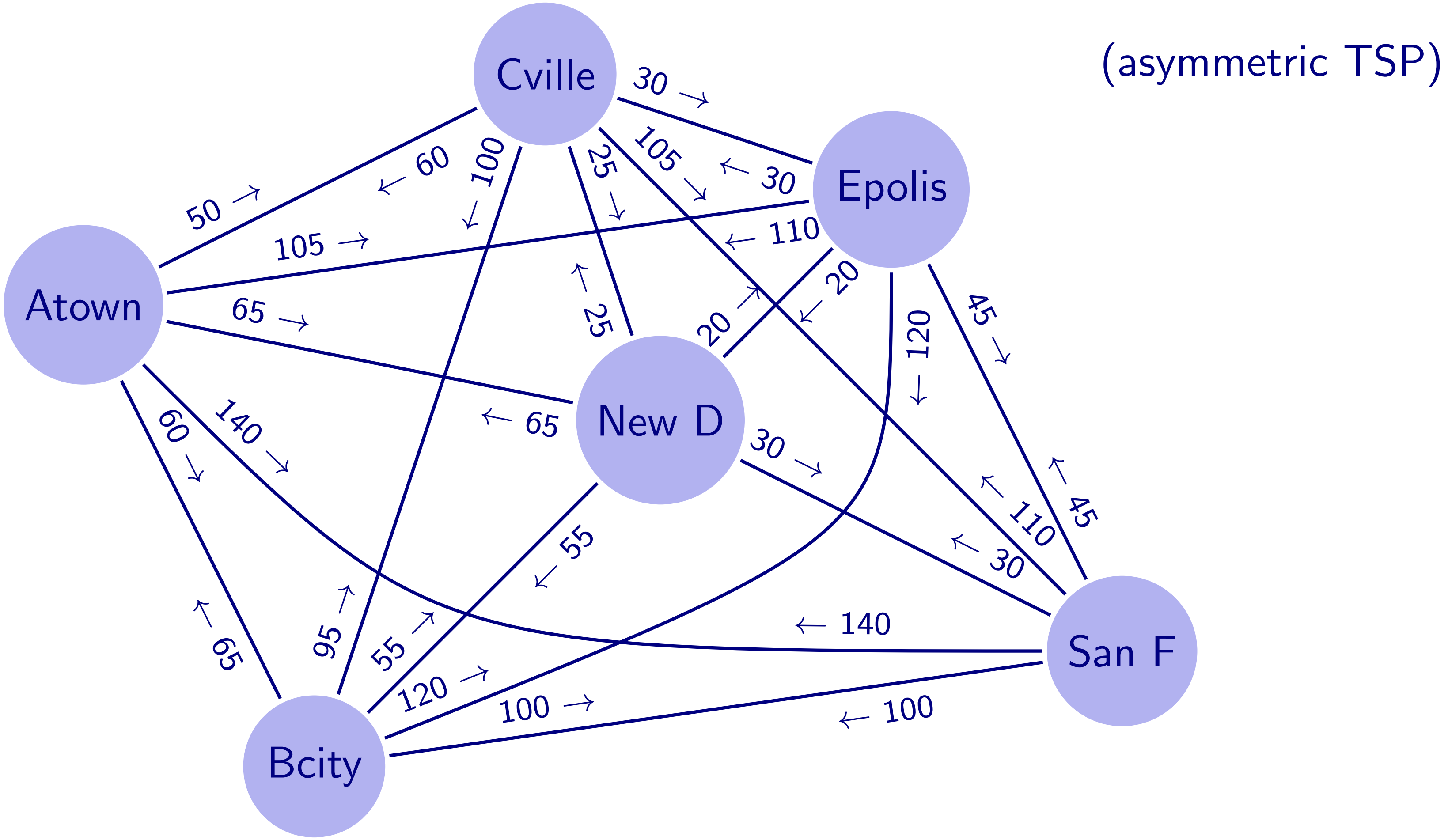
Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



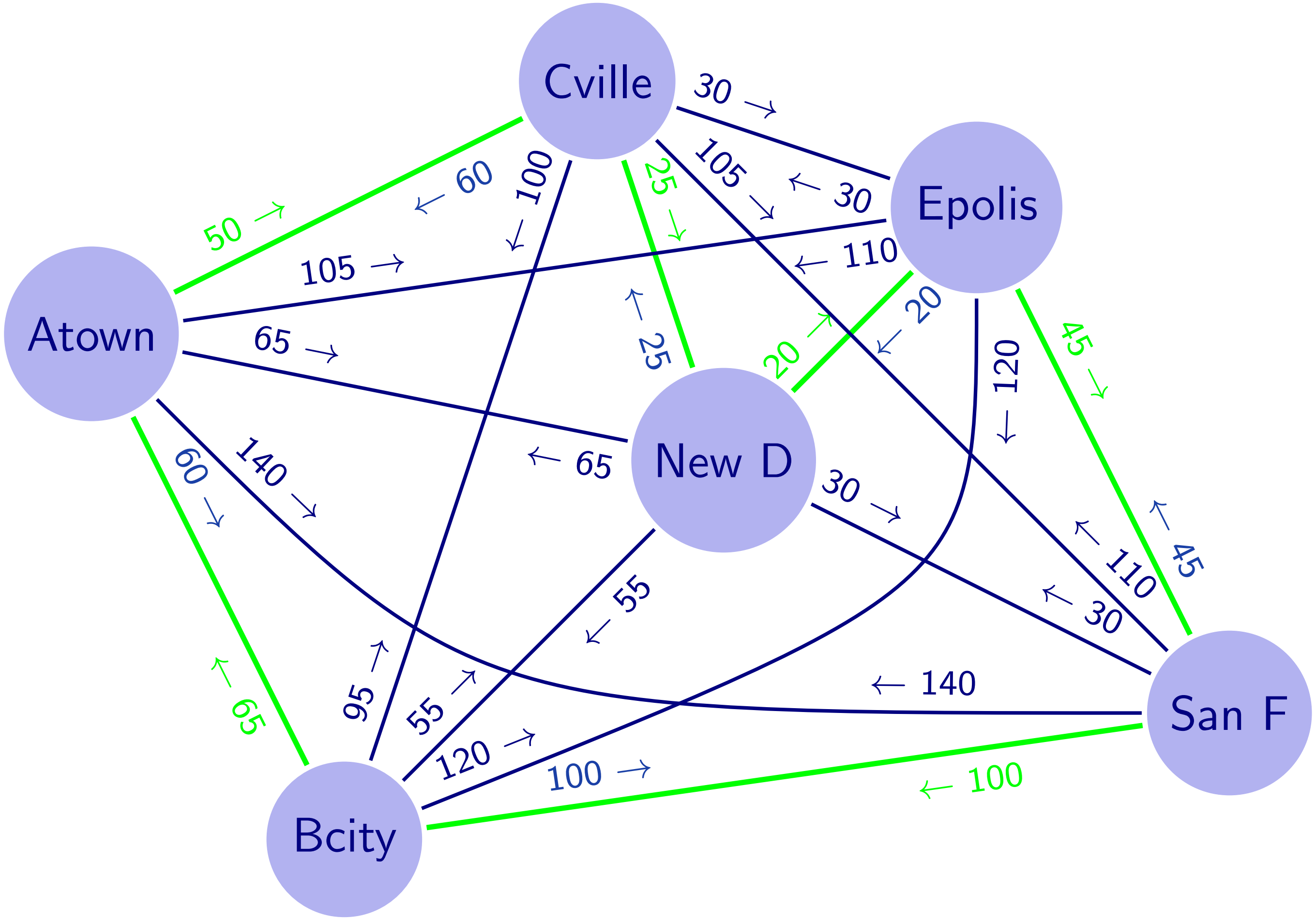
Problem formulation: Given a set of cities together with travel times to travel from every city to every other, find a tour leading through every city such that the total travel time is minimized.



V $x_{i,j} \in \{0,1\}$ ➡ $x_{i,j} = 1 \Leftrightarrow$ tour uses route from city i to city j

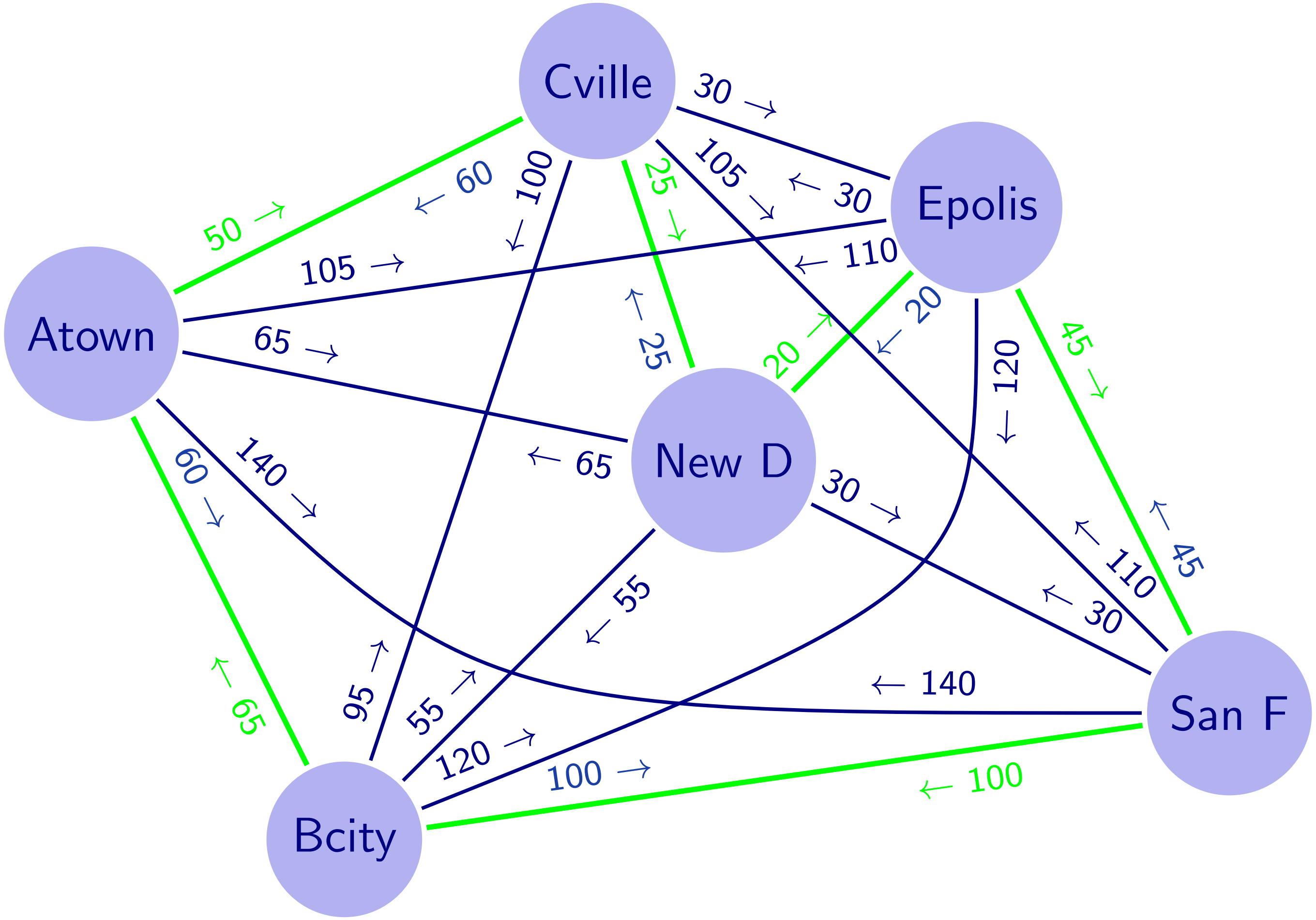
V $x_{i,j} \in \{0,1\}$ ➡ $x_{i,j} = 1 \Leftrightarrow$ tour uses route from city i to city j

▷ Example:



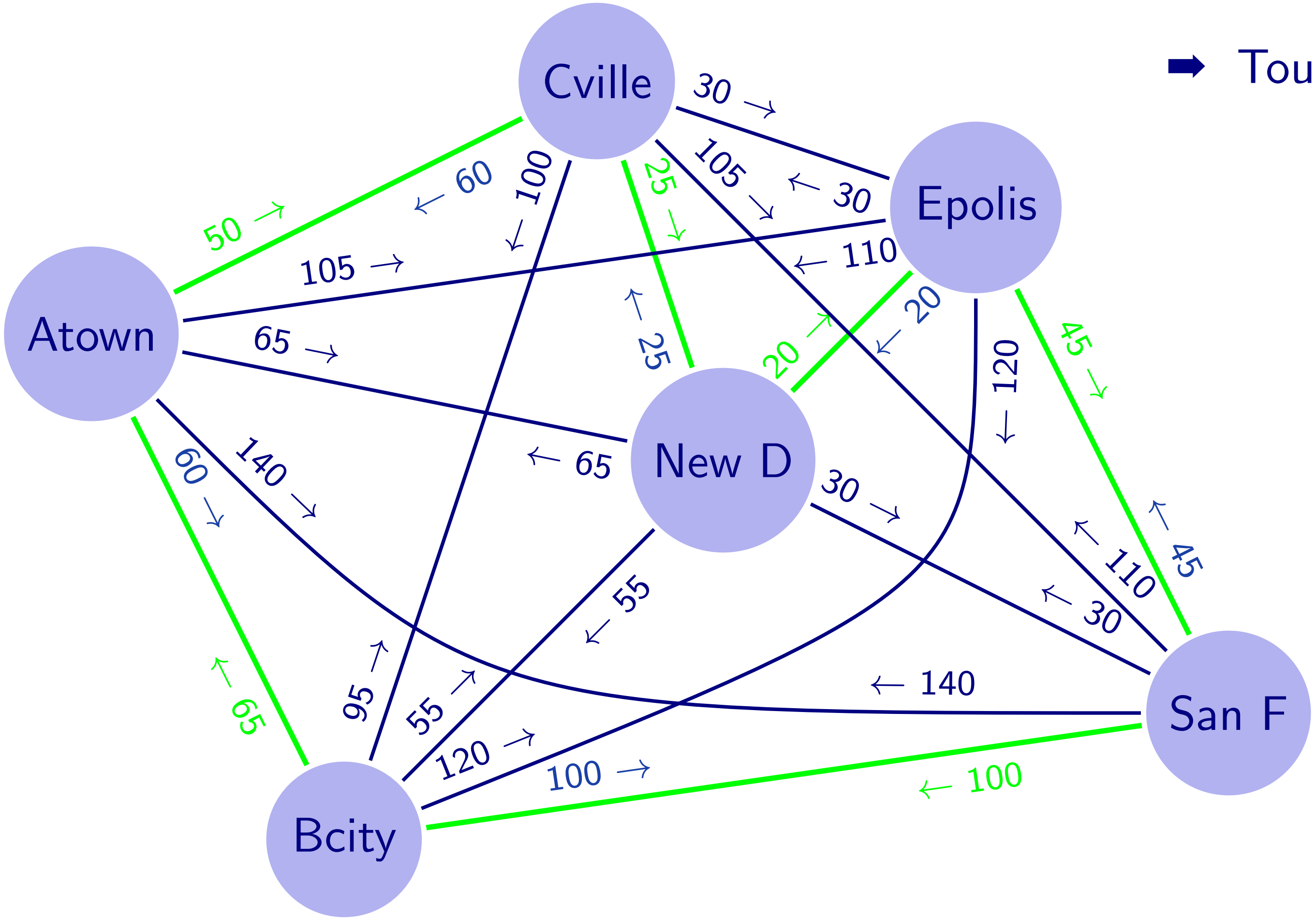
V $x_{i,j} \in \{0,1\}$ ➡ $x_{i,j} = 1 \Leftrightarrow$ tour uses route from city i to city j

▷ Example: $x_{\text{A,C}} = x_{\text{C,D}} = x_{\text{D,E}} = x_{\text{E,F}} = x_{\text{F,B}} = x_{\text{B,A}} = 1$, all other variables $= 0$



V $x_{i,j} \in \{0,1\}$ ➡ $x_{i,j} = 1 \Leftrightarrow$ tour uses route from city i to city j

▷ Example: $x_{\text{A,C}} = x_{\text{C,D}} = x_{\text{D,E}} = x_{\text{E,F}} = x_{\text{F,B}} = x_{\text{B,A}} = 1$, all other variables $= 0$



➡ Tour length: 305

S Cities: $C = \{1, \ldots, n\}$

V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i, j \in C$, $i \neq j$

S Cities: $C = \{1, \ldots, n\}$

Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i, j \in C$, $i \neq j$

S Cities: $C = \{1, \ldots, n\}$

Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

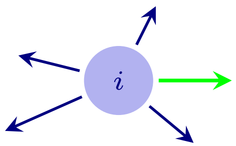
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i, j \in C,\ i \neq j$

S Cities: $C = \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C,\ i \neq j$

Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



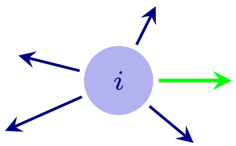
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i,j \in C$, $i \neq j$

S Cities: $C = \{1,\ldots,n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i,j \in C$, $i \neq j$

Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



$$\sum_{j \neq i} x_{i,j} = 1 \quad \text{for all cities } i \in C$$

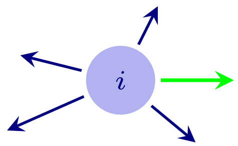
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i, j \in C$, $i \neq j$

S Cities: $C = \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C$, $i \neq j$

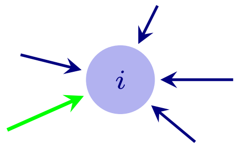
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



$\sum_{j \neq i} x_{i,j} \;=\; 1$ for all cities $i \in C$

(city inbound)



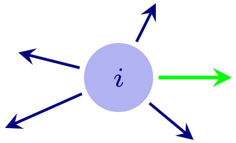
V (usage of routes) $x_{i,j} \;\in\; \{0, 1\}$ for all $i, j \in C,\ i \neq j$

S Cities: $C \;=\; \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C,\ i \neq j$

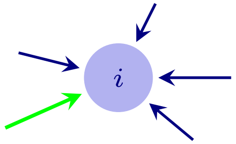
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



$\sum_{j \neq i} x_{i,j} = 1$ for all cities $i \in C$

(city inbound)



$\sum_{j \neq i} x_{j,i} = 1$ for all cities $i \in C$

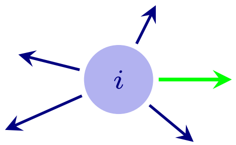
V (usage of routes) $x_{i,j} \in \{0, 1\}$ for all $i, j \in C$, $i \neq j$

S Cities: $C = \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C$, $i \neq j$

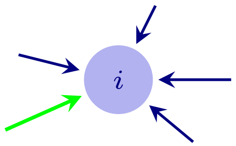
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



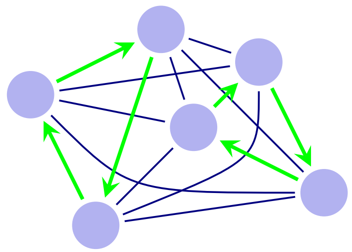
$\sum_{j \neq i} x_{i,j} = 1$ for all cities $i \in C$

(city inbound)



$\sum_{j \neq i} x_{j,i} = 1$ for all cities $i \in C$

(one cycle!)



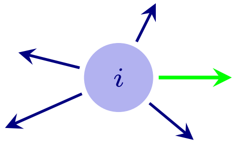
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i,j \in C$, $i \neq j$

S Cities: $C = \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i,j \in C$, $i \neq j$

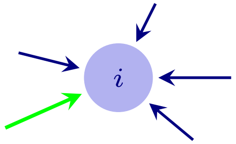
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



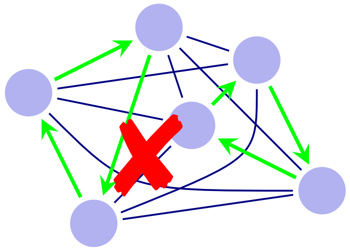
$\sum_{j \neq i} x_{i,j} = 1$ for all cities $i \in C$

(city inbound)



$\sum_{j \neq i} x_{j,i} = 1$ for all cities $i \in C$

(one cycle!)



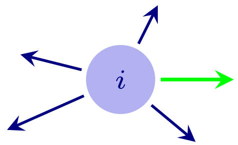
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i,j \in C$, $i \neq j$

S Cities: $C = \{1,\ldots,n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i,j \in C$, $i \neq j$

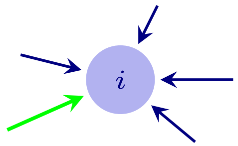
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



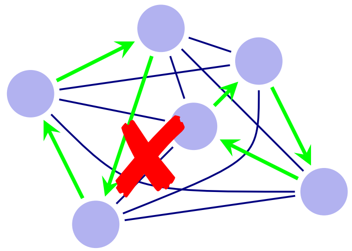
$$\sum_{j \neq i} x_{i,j} = 1 \quad \text{for all cities } i \in C$$

(city inbound)



$$\sum_{j \neq i} x_{j,i} = 1 \quad \text{for all cities } i \in C$$

(one cycle!)

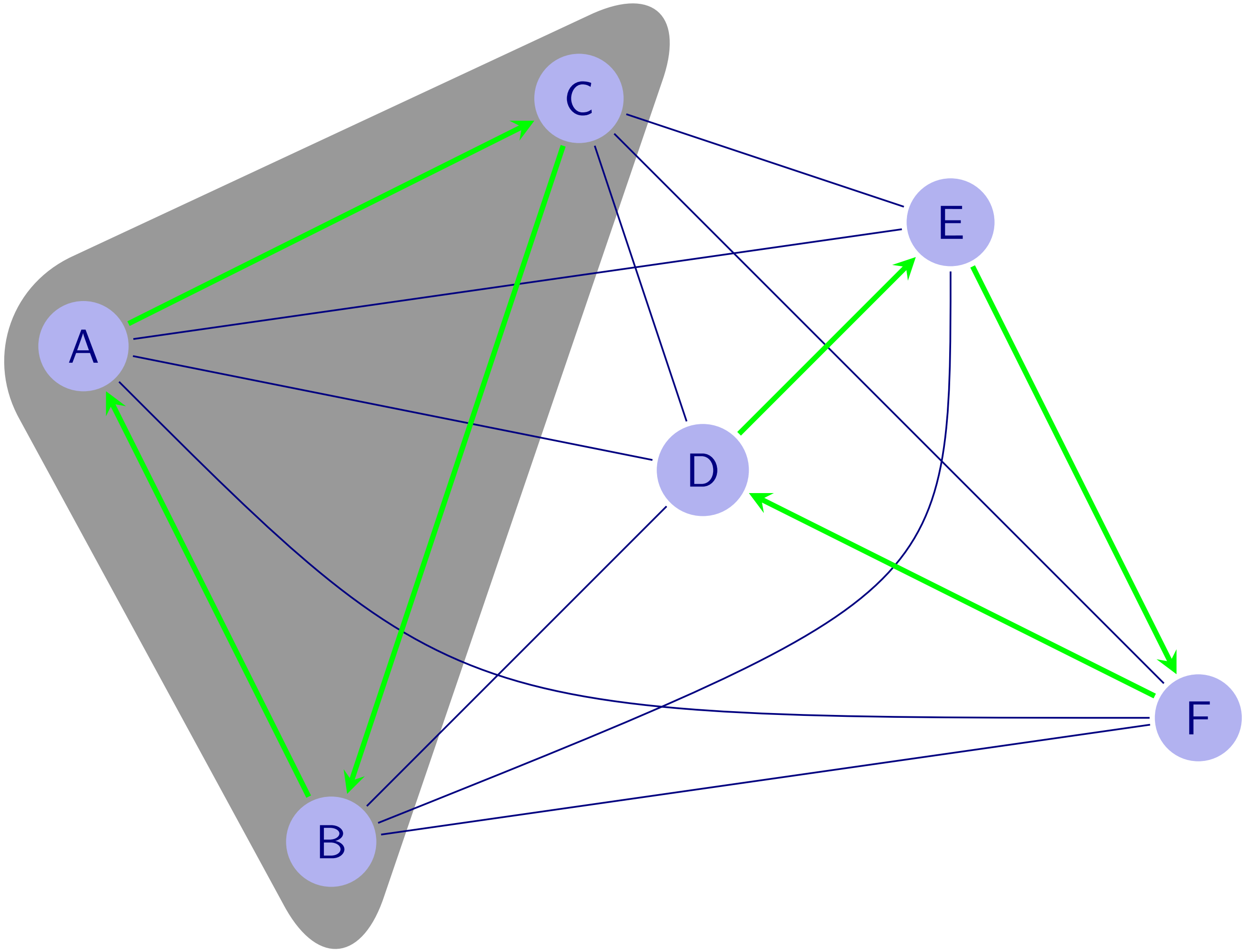


➡ **?**

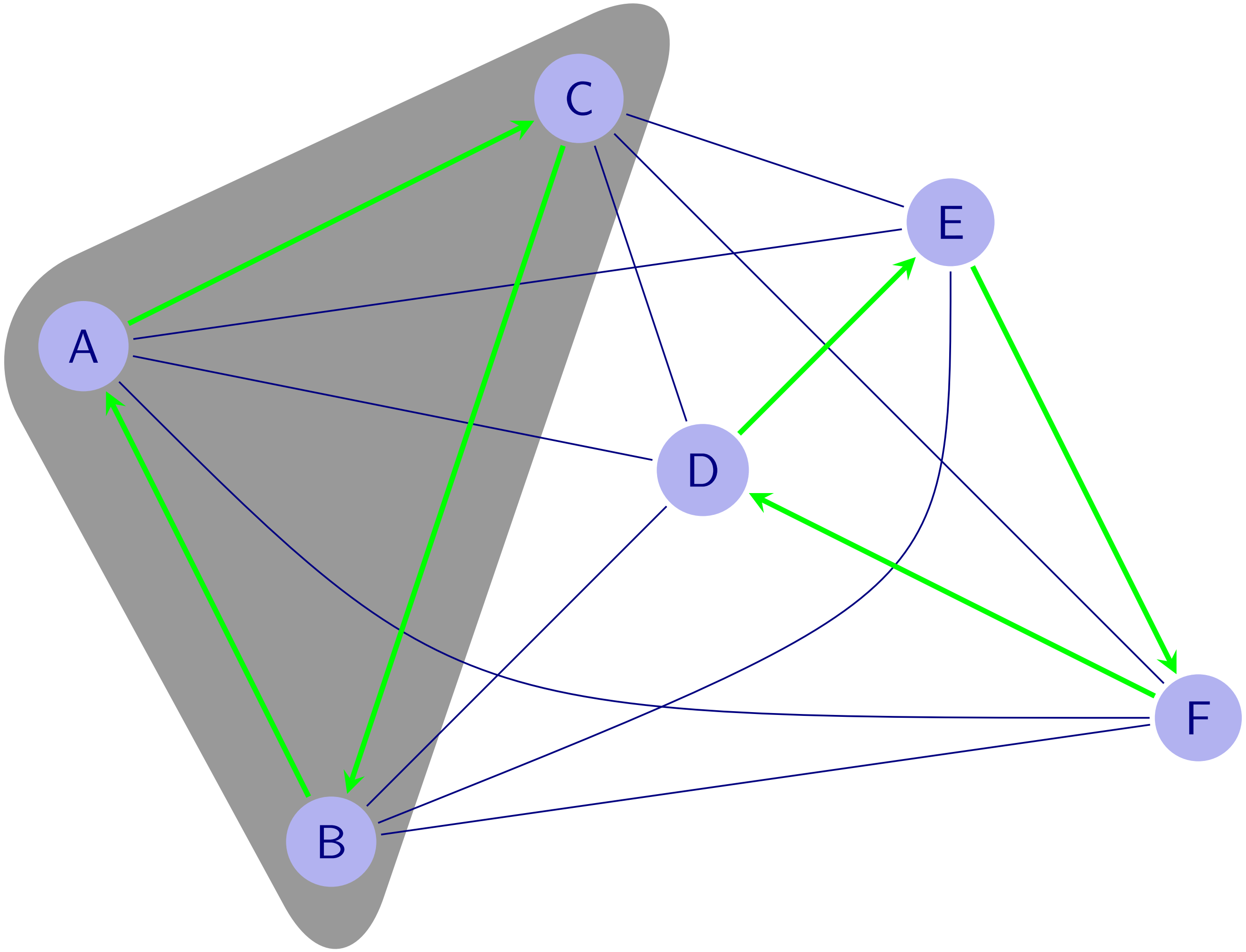
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i,j \in C$, $i \neq j$

S Cities: $C = \{1,\ldots,n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i,j \in C$, $i \neq j$

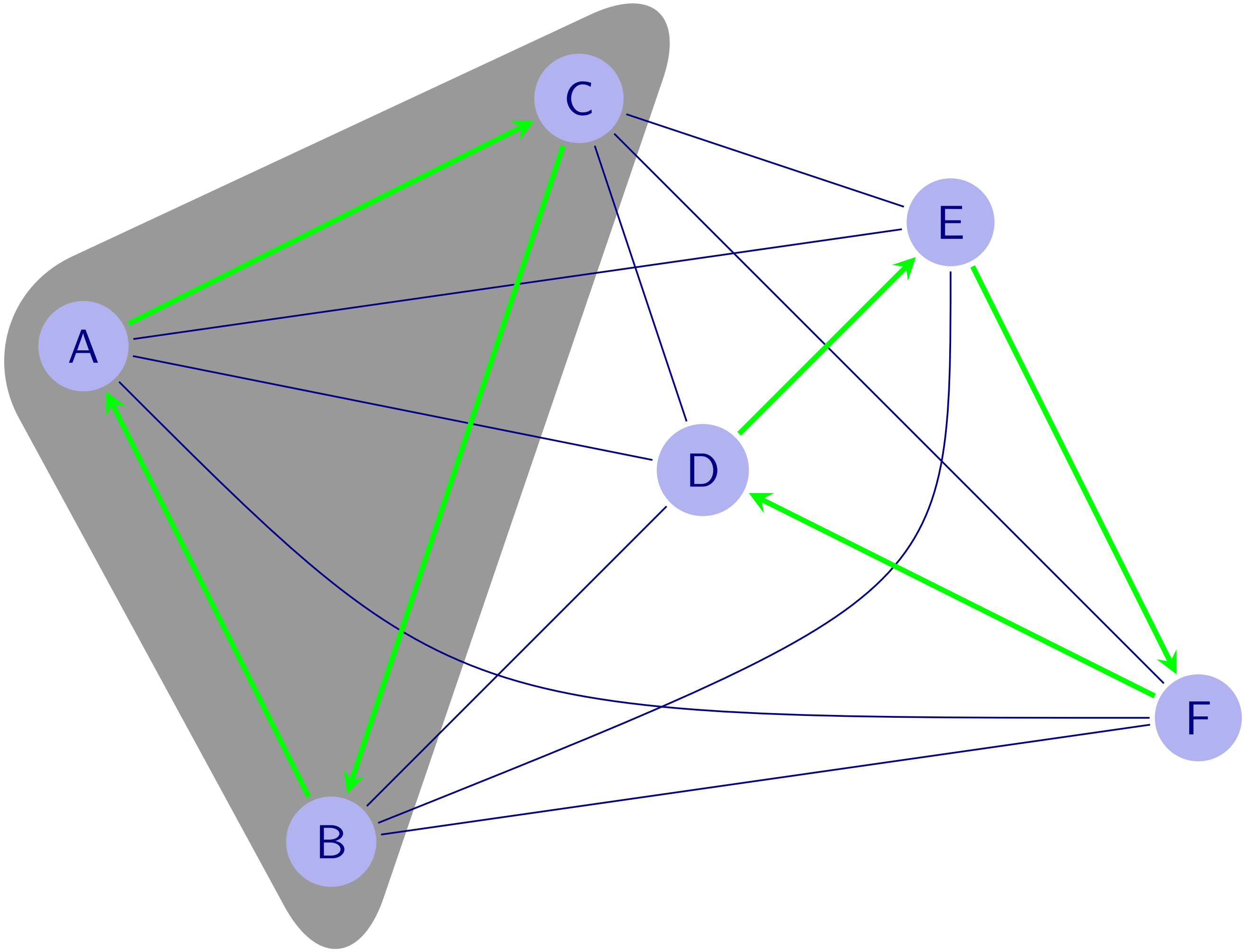

C
E
A
D
F
B

▷ Force the tour to leave every subset of cities at least once



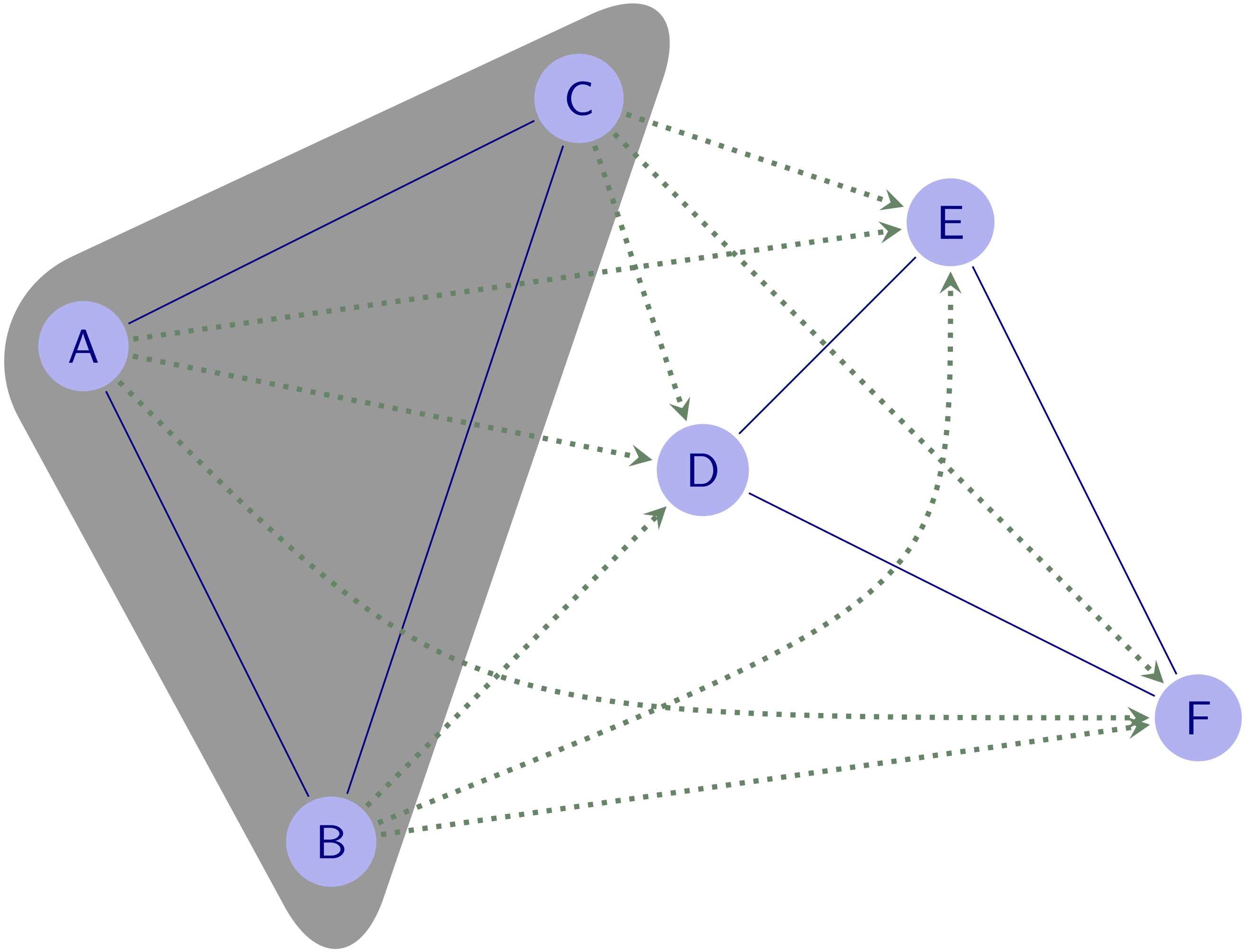
▷ Force the tour to leave every subset of cities at least once

➡ $$\sum_{i \in S,\, j \notin S} x_{i,j} \geq 1 \quad \text{for all subsets } S \subset C,\ \emptyset \neq S \neq C$$



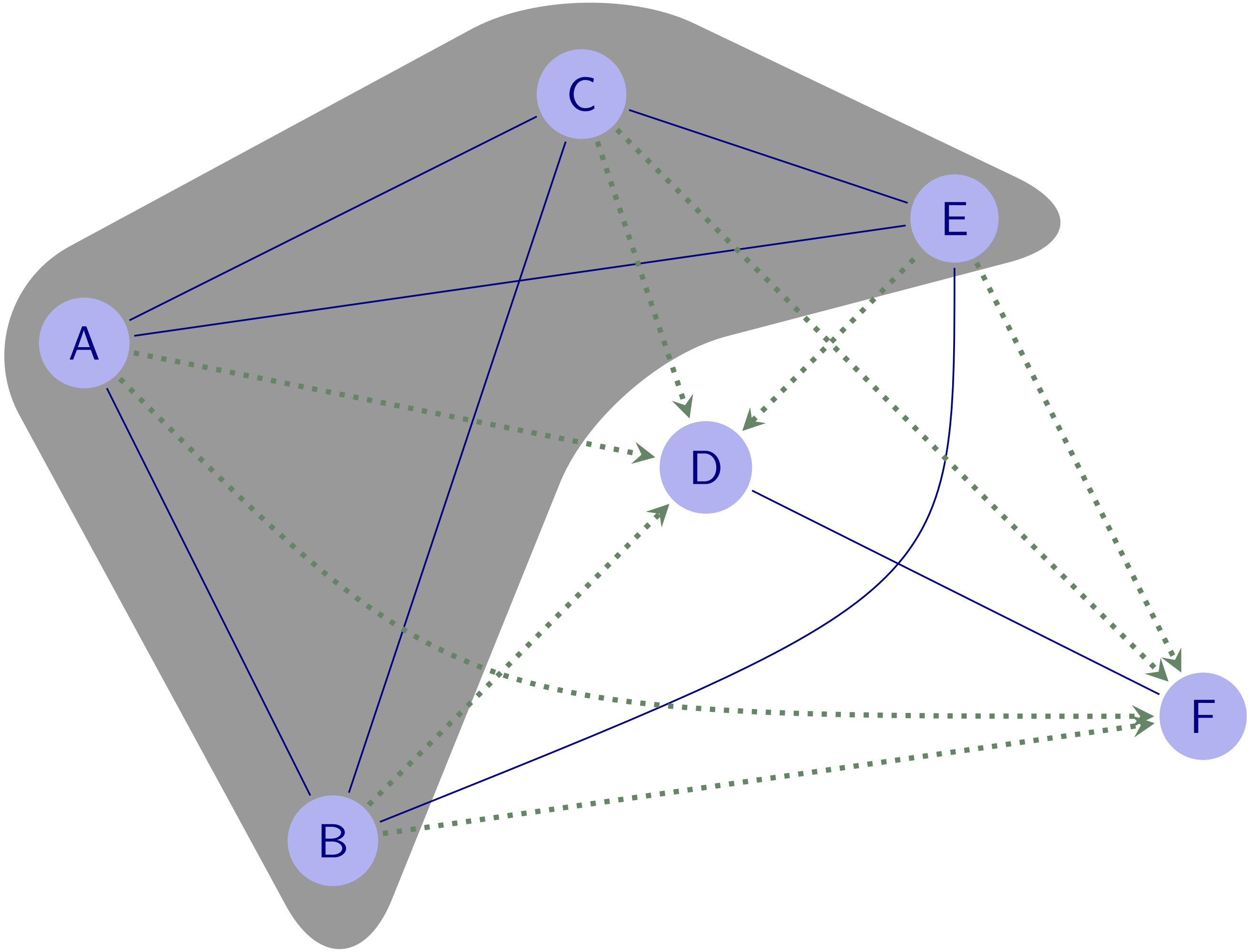
- ▷ Force the tour to leave every subset of cities at least once

➡ $$\sum_{i \in S,\, j \notin S} x_{i,j} \geq 1 \quad \text{for all subsets } S \subset C,\ \emptyset \neq S \neq C$$



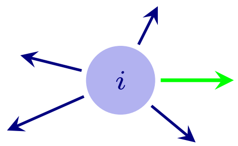
- Force the tour to leave every subset of cities at least once

➡ $$\sum_{i \in S,\, j \notin S} x_{i,j} \geq 1 \quad \text{for all subsets } S \subset C,\ \emptyset \neq S \neq C$$



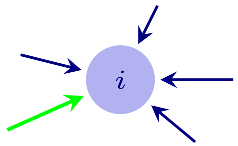
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



$$\sum_{j \neq i} x_{i,j} = 1 \quad \text{for all cities } i \in C$$

(city inbound)



$$\sum_{j \neq i} x_{j,i} = 1 \quad \text{for all cities } i \in C$$

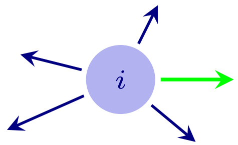
V (usage of routes) $x_{i,j} \in \{0,1\}$ for all $i, j \in C$, $i \neq j$

S Cities: $C = \{1, \dots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C$, $i \neq j$

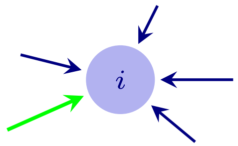
Objective: minimize (total tour length) $\sum_{i,j \in C,\, i \neq j} \ell_{i,j}\, x_{i,j}$

C (city outbound)



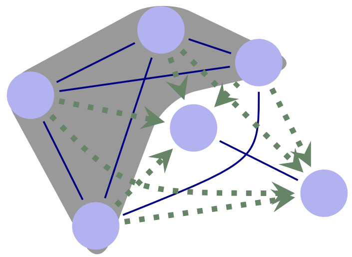
$\sum_{j \neq i} x_{i,j} = 1$ for all cities $i \in C$

(city inbound)



$\sum_{j \neq i} x_{j,i} = 1$ for all cities $i \in C$

(one cycle!)



$\sum_{i \in S,\, j \notin S} x_{i,j} \geq 1$ for all subsets of cities $S \subset C,\ \emptyset \neq S \neq C$

V (usage of routes) $x_{i,j} \in \{0, 1\}$ for all $i, j \in C,\ i \neq j$

S Cities: $C = \{1, \ldots, n\}$

P Route lengths: $\ell_{i,j} \geq 0$ for all $i, j \in C,\ i \neq j$

- Problem: the model still has exponential size!

▷ Problem: the model still has exponential size!

# cities	# possible tours	# constraints
5	24	40
10	362,880	1,042
15	87,178,291,200	32,796
20	121,645,100,408,832,000	1,048,614
$\vdots$		
n	$(n-1)!$	$2^n - 2 + 2n$

▷ Problem: the model still has exponential size!

# cities	# possible tours	# constraints
5	24	40
10	362,880	1,042
15	87,178,291,200	32,796
20	121,645,100,408,832,000	1,048,614
⋮		
n	$(n-1)!$	$2^n - 2 + 2n$

➡ The IP formulation gives optimal solutions only for relatively small input size (# cities $<$ 20)

▷ Problem: the model still has exponential size!

# cities	# possible tours	# constraints
5	24	40
10	362,880	1,042
15	87,178,291,200	32,796
20	121,645,100,408,832,000	1,048,614
⋮		
n	$(n-1)!$	$2^n - 2 + 2n$

➡ The IP formulation gives optimal solutions only for relatively small input size (# cities $<$ 20)

➡ To solve the TSP for larger instances

- try to find solutions by using heuristics or approximation algorithms

▷ Problem: the model still has exponential size!

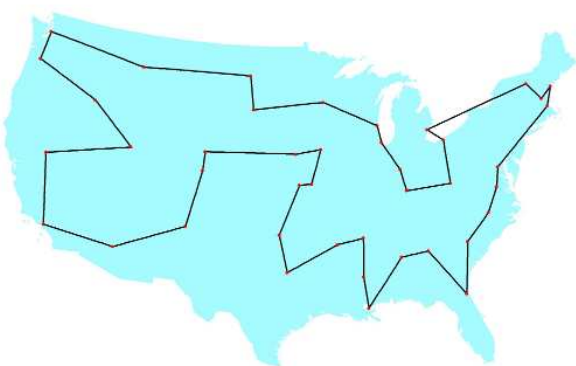
# cities	# possible tours	# constraints
5	24	40
10	362,880	1,042
15	87,178,291,200	32,796
20	121,645,100,408,832,000	1,048,614
⋮		
n	$(n-1)!$	$2^n - 2 + 2n$

➡ The IP formulation gives optimal solutions only for relatively small input size (# cities $<$ 20)

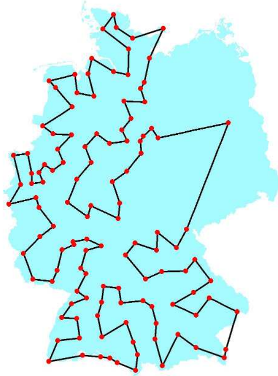
➡ To solve the TSP for larger instances

- try to find solutions by using heuristics or approximation algorithms
- use lower bounds provided by the IP model to estimate quality of solutions

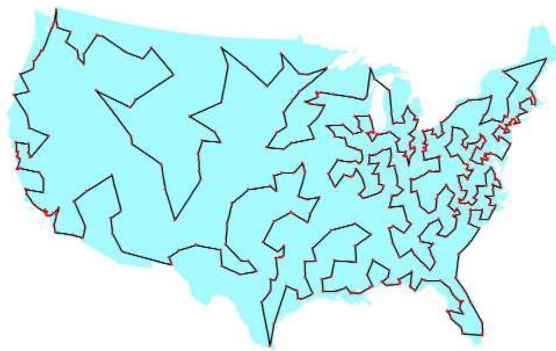
TSP – optimal solution gallery



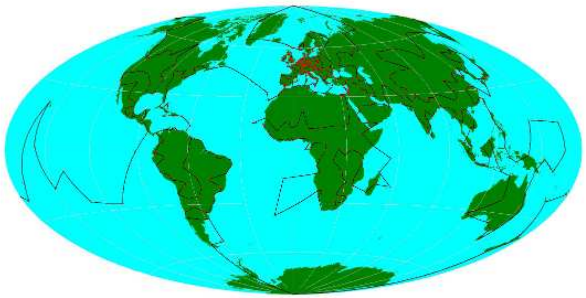
1954: Dantzig et al: 49 cities



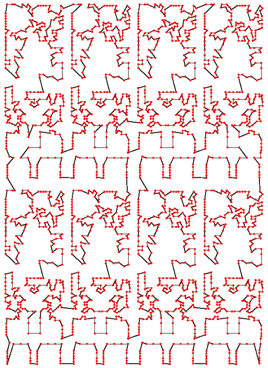
1977: Grötschel: 120 cities



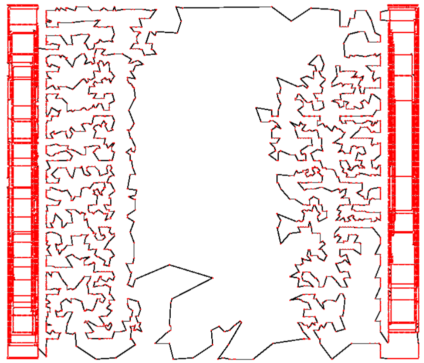
1987: Padberg, Rinaldi: 532 cities



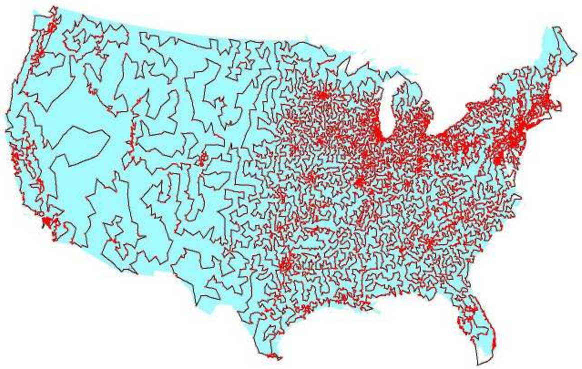
1987: Grötschel. Holland: 666 cities



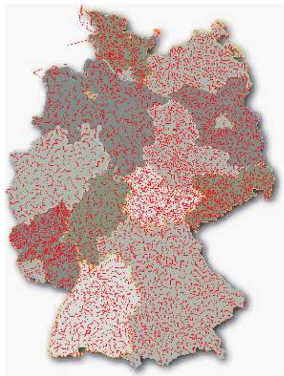
1987: Padberg, Rinaldi: 2392 “cities”



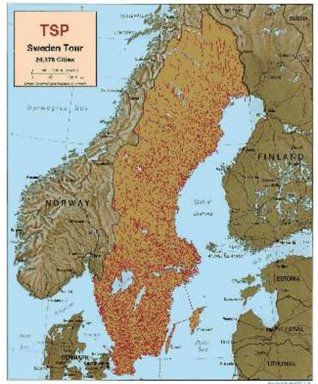
1994: Applegate et al: 7397 “cities”



1998: Applegate et al: 13509 cities



2001: Applegate et al: 15112 cities



2004: Applegate et al: 24978 cities

- ▷ Models, Data and Algorithms
- ▷ Linear Optimization
- ▷ Mathematical Background: Polyhedra, Simplex-Algorithm
- ▷ Sensitivity Analysis; (Mixed) Integer Programming
- ▷ MIP Modelling
- ▷ MIP Modelling: More Examples; Branch & Bound
- ▷ Cutting Planes; Combinatorial Optimization: Examples, Graphs, Algorithms
- ▷ TSP-Heuristics
- ▷ Complexity Theory
- ▷ Nonlinear Optimization
- ▷ Scheduling
- ▷ Lot Sizing
- ▷ Multicriteria Optimization
- ▷ Oral exam