

# Algorithms in the Real World

## Linear and Integer Programming III

### - Integer Programming

- Applications
- Algorithms

# Integer (linear) Programming

**minimize:**  $c^T x$

**subject to:**  $Ax \leq b$

$x \geq 0$

$x \in \mathbb{Z}^n$

## Related Problems

- Mixed Integer Programming (MIP)
- Zero-one programming
- Integer quadratic programming
- Integer nonlinear programming

# History

- Introduced in 1951 (Dantzig)
- TSP as special case in 1954 (Dantzig)
- First convergent algorithm in 1958 (Gomory)
- General branch-and-bound technique 1960 (Land and Doig)
- Frequently used to prove bounds on approximation algorithms (late 90s)

# Current Status

- Has become “dominant” over linear
- Saves industry billions of dollars/year
- Can solve 10,000+ city TSP problems
- 1 million variable LP approximations
- Branch-and-bound, cutting plane, and separation all used in practice
- General purpose packages do not tend to work as well as with linear programming -- knowledge of the domain is critical.

# Subproblems/Applications

- **Facility location**  
Locating warehouses or franchises (e.g., a Burger King)
- **Set covering and partitioning**  
Scheduling airline crews
- **Multicommodity distribution**  
Distributing auto parts
- **Traveling salesman and extensions**  
Routing deliveries
- **Capital budgeting**
- **Other applications**  
VLSI layout, clustering

# Knapsack Problem

## Integer (zero-one) Program:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to:} & ax \leq b \\ & x \text{ binary} \end{array}$$

## where:

$b$  = maximum weight

$c_i$  = utility of item  $i$

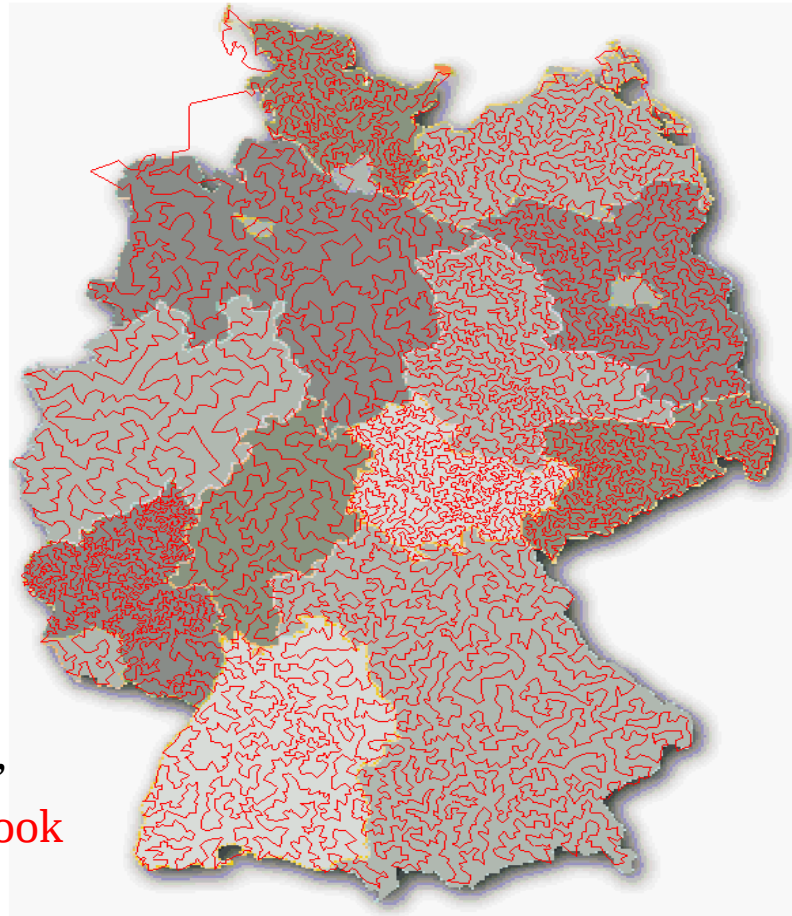
$a_i$  = weight of item  $i$

$x_i = 1$  if item  $i$  is selected, or 0 otherwise

The problem is NP-hard.

# Traveling Salesman Problem

Find shortest tours that visit all of  $n$  cities.



courtesy: Applegate,  
Bixby, Chvátal, and Cook

# Traveling Salesman Problem

**minimize:** 
$$\sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

**subject to:** 
$$\sum_{j=0}^n x_{ij} = 2 \quad 1 \leq i \leq n \quad (\text{path enters and leaves})$$

$$x_{ji} = x_{ij}, \text{ binary}$$

$c_{ij} = c_{ji}$  = distance from city  $i$  to city  $j$   
(assuming **symmetric version**)

$x_{ij}$  if tour goes from  $i$  to  $j$  or  $j$  to  $i$ , and 0 otherwise

**Anything missing?**



# Traveling Salesman Problem

**minimize:** 
$$\sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

**subject to:** 
$$\sum_{j=1}^n x_{ij} = 1 \quad 1 \leq i \leq n \quad (\text{out degrees} = 1)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad 1 \leq j \leq n \quad (\text{in degrees} = 1)$$

$$t_i - t_j + nx_{ij} \leq n - 1 \quad 2 \leq i, j \leq n \quad (??)$$

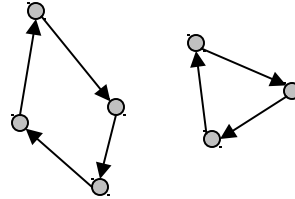
$c_{ij}$  = distance from city  $i$  to city  $j$

$x_{ij}$  = 1 if tour visits  $i$  then  $j$ , and 0 otherwise (binary)

$t_i$  = “potential” of node  $i$ , real number to solve for

# Traveling Salesman Problem

The last set of constraints:  $t_i - t_j + nx_{ij} \leq n - 1 \quad 2 \leq i, j \leq n$   
prevents “subtours”:



Consider a cycle that goes from some node 4 to 5,

$$t_4 - t_5 + nx_{4,5} \leq n-1 \Rightarrow t_5 \geq t_4 + 1 + n(x_{4,5}-1) \Rightarrow t_5 \geq t_4 + 1$$

Thus,  $t$  has to increase by at least 1 along each edge of the cycle that does not include vertex 1.

(On the other hand, if  $x_{4,5}=0$ , we have  $t_4 - t_5 \leq n-1$ , which just limits distance from vertex 5 to 4 to at most  $n-1$ .)

Therefore, if a cycle of length  $m$  contains vertex 4 but does not go through vertex 1,  $t_4 \geq t_4 + m$ , a contradiction.

**Every cycle must go through vertex 1.**

# Traveling Salesman Problem

Many “Real World” **applications** based on the TSP.

- They typically involve more involved constraints
- Not just routing type problems.

Consider a drug company with  $k$  drugs they can make at a lab. They can only make the drugs one at a time. The cost of converting the equipment from making drug  $i$  to drug  $j$  is  $c_{ij}$

Best solutions are based on IP (but not MTZ formulation)

- Applegate, Bixby, et. al., have solutions for more than 15K cities in Germany  
> 150,000 CPU hours (**more info**)
- Involves “branch-and-bound” and “cutting planes”

# Set Covering Problem

Find cheapest sets that cover all elements



Courtesy: Darmstadt  
University of Technology

# Set Covering and Partitioning

**Given m sets and n items:**

$$A_{ij} = \begin{cases} 1, & \text{if set } j \text{ includes item } i \\ 0, & \text{otherwise} \end{cases}$$

$$c_j = \text{cost of set } j$$

$$x_j = \begin{cases} 1, & \text{if set } j \text{ is included} \\ 0, & \text{otherwise} \end{cases}$$

Columns = sets  
Rows = items

**Set covering:**

**minimize:**  $c^T x$

**subject to:**  $Ax \geq 1, x \text{ binary}$

**Set partitioning:**

**minimize:**  $c^T x$

**subject to:**  $Ax = 1, x \text{ binary}$

# Set Covering and Partitioning

set	member s	cost
$s_1$	{a,c,d}	.5
$s_2$	{b,c}	.2
$s_3$	{b,e}	.3
$s_4$	{a,d}	.1
$s_5$	{c,e}	.2
$s_6$	{b,c,e}	.6
$s_7$	{c,d}	.2

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

**Best cover:**  $s_2, s_4, s_5 = .5$

**Best partition:**  $s_4, s_6 = .7$

# Set Covering and Partitioning

## Applications:

- **Facility location.**

Each set is a facility (e.g., warehouse, fire station, emergency response center).

Each item is an area that needs to be covered

- **Crew scheduling.**

Each set is a route for a particular crew member (e.g. NYC->Pit->Atlanta->NYC).

Each item is a flight that needs to be covered.

# Constraints Expressible with IP

Many constraints are expressible with integer programming:

- logical constraints (e.g.,  $x$  implies not  $y$ )
- $k$  out of  $n$
- piecewise linear functions
- ... and many more



# Constraints Expressible with IP

## Logical constraints ( $x_1, x_2$ binary):

$$\text{Either } x_1 \text{ or } x_2 \quad \Rightarrow \quad -x_1 - x_2 \leq -1$$

$$\text{If } x_1 \text{ then } x_2 \quad \Rightarrow \quad x_1 - x_2 \leq 0$$

$$k \text{ out of } n \quad \Rightarrow \quad \sum_{1 \leq i \leq n} x_i = k$$

## Combining constraints:

$$\text{Either } a_1x \leq b_1 \text{ or } a_2x \leq b_2 \quad \Rightarrow \quad \begin{array}{l} a_1x - My \leq b_1 \\ a_2x - M(1-y) \leq b_2 \end{array}$$

$y$  is a binary variable,  $M$  needs to be “large”,  
 $a_1, a_2$ , and  $x$  can be vectors

# Constraints Expressible with IP

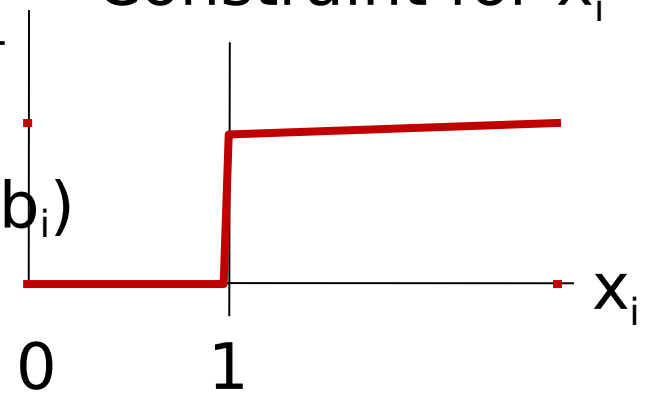
- **Discrete variables:**  $x_i$  in  $\{k_1, k_2, \dots, k_n\}$ 
  - Create new binary vars  $z_j$  and add constraints

$$x_i = \sum_{1 \leq i \leq n} z_i k_i \quad \text{and} \quad \sum_{1 \leq i \leq n} z_i = 1$$

- **Piecewise linear functions:**

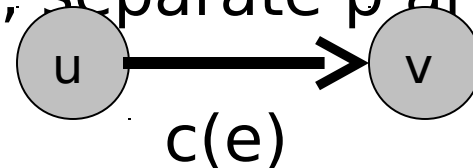
- If  $x_i \geq 1$  then  $a_i x_i \leq b_i$
- Convert to  $(x_i < 1)$  or  $(a_i x_i \leq b_i)$   
and use method from  
previous page

Constraint for  $x_i$



# Tricks for Expressing Constraints

- **Covering Constraints:**  $Ax \geq b$  (non-negative  $a_{ij}$ 's)
- **Packing constraints:**  $Ax \leq b$  (non-negative  $a_{ij}$ 's)
- **Connectivity constraints** (e.g., for network design):
  - flow formulation: to connect  $s$  and  $t$ , buy enough edges to support a unit  $s$ - $t$  flow. Variables  $d, x$ . Edge costs/lengths  $c(e)$ :
- **Cut constraints** (e.g., for clustering):
  - Distance formulation: e.g., separate  $p$  and  $q$ 
    - $d(p, v) \leq d(p, u) + c(e)$  for each edge  $e=(u, v)$
    - $d(p, v) \geq 0$  for each  $v$
    - $d(p, q) \geq \infty$



# Algorithms

1. Use a linear program
  - round to integer solution (what if not feasible?)
2. Search
  - Branch and bound (integer ranges)
  - Implicit (0-1 variables)
3. Cutting planes
  - Many variants

## Relationships between LP and IP Solutions

- LP solution is an upper bound on IP solution (assuming maximization)
- If LP is infeasible then IP is infeasible
- If LP solution is integral (all variables have integer values), then it is the IP solution.

# Linear Programming Solution

1. Some LP problems will always have integer solutions
  - transportation problem (send finished goods from factories to customers)
  - assignment problem (assign workers to jobs)
  - min-cost network flow (w/ integer capacities)

These are problems with a unimodular matrix  $A$  (integer matrix invertible over the integers,  $\det(A) = \pm 1$ ).

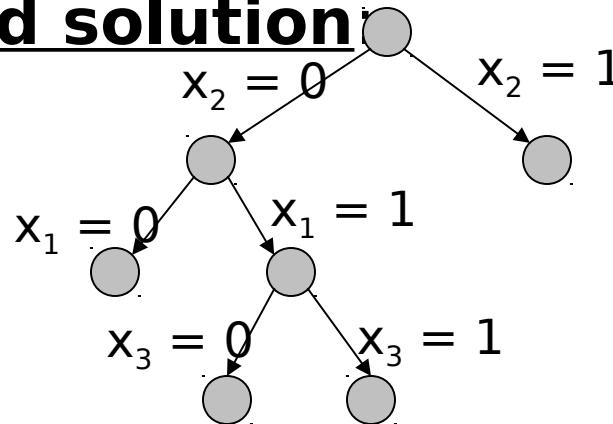
2. Solve as linear program and round. Can violate constraints, and be non-optimal. Works OK if
  - integer variables take on large values
  - accuracy of constraints is questionable

# Branch and Bound

Lets first consider **0-1** programs.

**Exponential solution**: try all  $\{0,1\}^n$

**Branch-and-bound solution**:



Traverse tree keeping current best solution.

If it can be shown that a subtree never improves on the current solution, **or** is infeasible, **prune** it.

# Zero-One Branch and Bound

**minimize:**  $z = c^T x$ , subject to:  $Ax \leq b$ ,  $x \geq 0$ ,  $x \in \{0,1\}^n$

Assume all elements of  $c$  are non-negative

**function**  $\mathbf{ZO}_r(A, b, c, x_f, z^*)$

//  $x_f$ : a fixed setting for a subset of the variables

//  $z^*$  is the cost of current best solution

$x = x_f + 0$  // set unconstrained variables to zero

**if** ( $c^T x \geq z^*$ ) **or** (no feasible completion of  $x_f$ ) **return**  $z^*$

**if** ( $Ax \leq b$ ) **then return**  $c^T x$

pick an unconstrained variable  $x_i$  from  $x$

$z_0^* = \mathbf{ZO}_r(A, b, x_f \cup \{x_i = 0\}, c, z^*)$

$z_1^* = \mathbf{ZO}_r(A, b, x_f \cup \{x_i = 1\}, c, z_0^*)$

**return**  $z_1^*$

**function**  $\mathbf{ZO}(A, b, c) = \mathbf{ZO}_r(A, b, c, \emptyset, \infty)$



# Zero-One Branch and Bound

**Checking for feasible completions:** check each constraint and find if minimum of left is greater than right.

**Example:**

$$x_f = \{x_1 = 1, x_3 = 0\}$$

and one of the constraints is

$$3x_1 + 2x_2 - x_3 + x_4 \leq 2$$

then

$$3 + 2x_2 - 0 + x_4 \leq 2$$

$$2x_2 + x_4 \leq -1$$

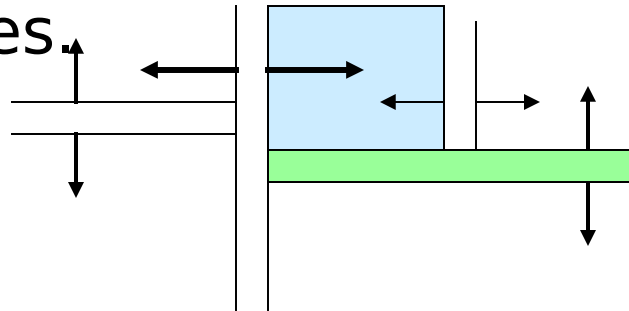
which is impossible.

# Integer Branch and Bound

The zero-one version is sometimes called “implicit enumeration” since it might enumerate all possibilities.

An integer version **cannot** branch on all possible integer values for a variable. Even if the integer range is bounded, it is not practical.

Will “bound” by adding inequalities to split the two branches



Since solutions are integral, each split can remove a strip of width 1 of non-integral settings for a variable

# Integer Branch and Bound

**maximize:**  $z = c^T x$  subject to:  $Ax \leq b, x \geq 0, x \in \mathbb{Z}^n$

**function**  $\mathbf{IP}_r(A_e, b_e, c, z^*)$

//  $A_e, b_e$  are  $A$  and  $b$  with additional constraints

//  $z^*$  is the cost of current best solution

$z, x, f = \text{LP}(A_e, b_e, c)$  //  $f$  indicates whether feasible

**if** not( $f$ ) **or** ( $z < z^*$ ) **return**  $z^*$

**if** (integer( $x$ )) **return**  $z$

pick a non-integer variable  $x_i'$  from  $x$

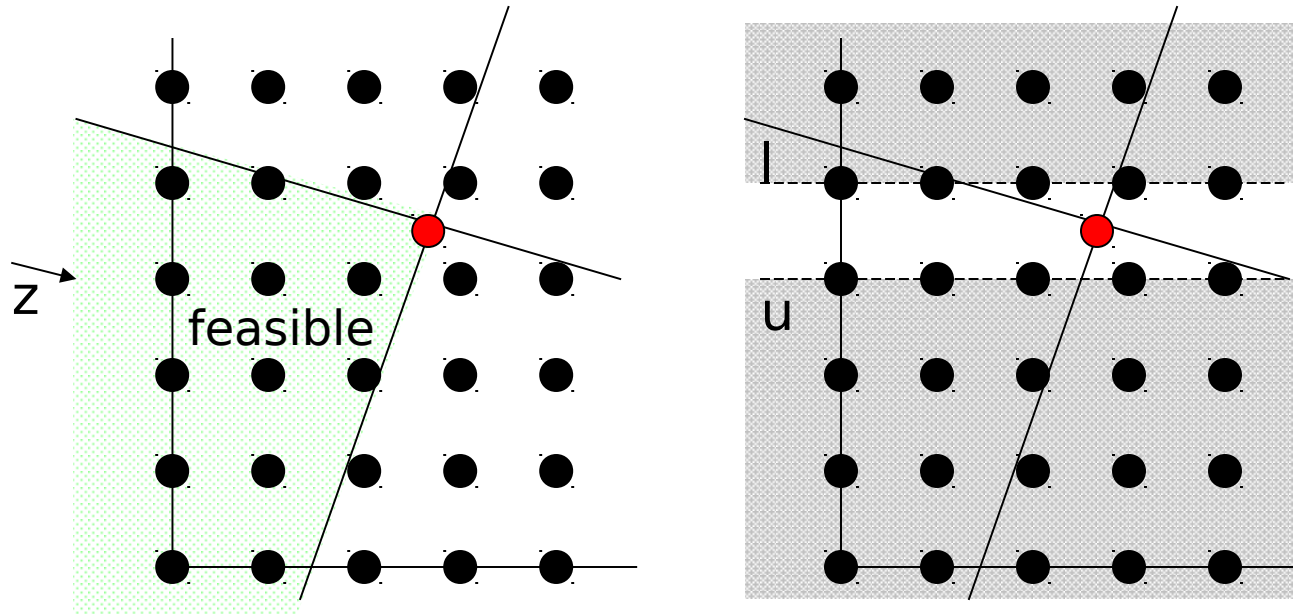
$z_l^* = \mathbf{IP}(\text{extend } A_e, b_e \text{ with } x_i \leq \lfloor x_i' \rfloor, c, z^*)$

$z_g^* = \mathbf{IP}(\text{extend } A_e, b_e \text{ with } -x_i \leq -\lceil x_i' \rceil, c, z_l^*)$

**return**  $z_g^*$

**function**  $\mathbf{IP}(A, b, c) = \mathbf{IP}_r(A, b, c, -\infty)$

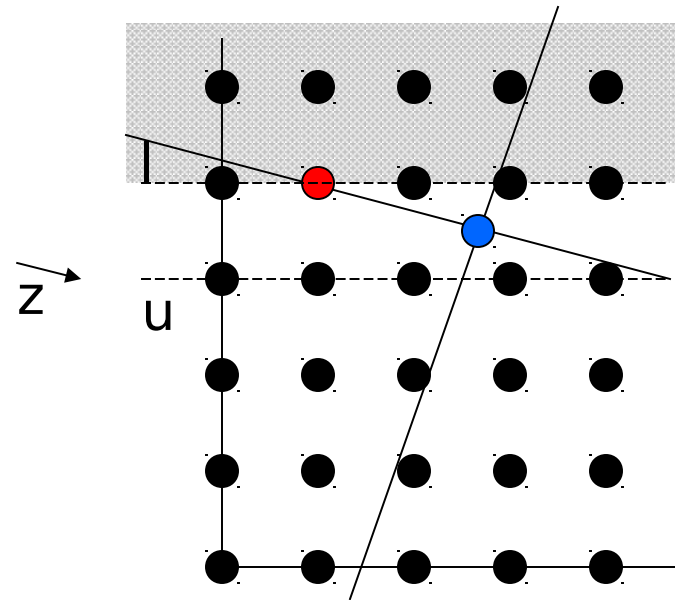
# Example



Find optimal solution.

Cut along  $y$  axis, and make two recursive calls

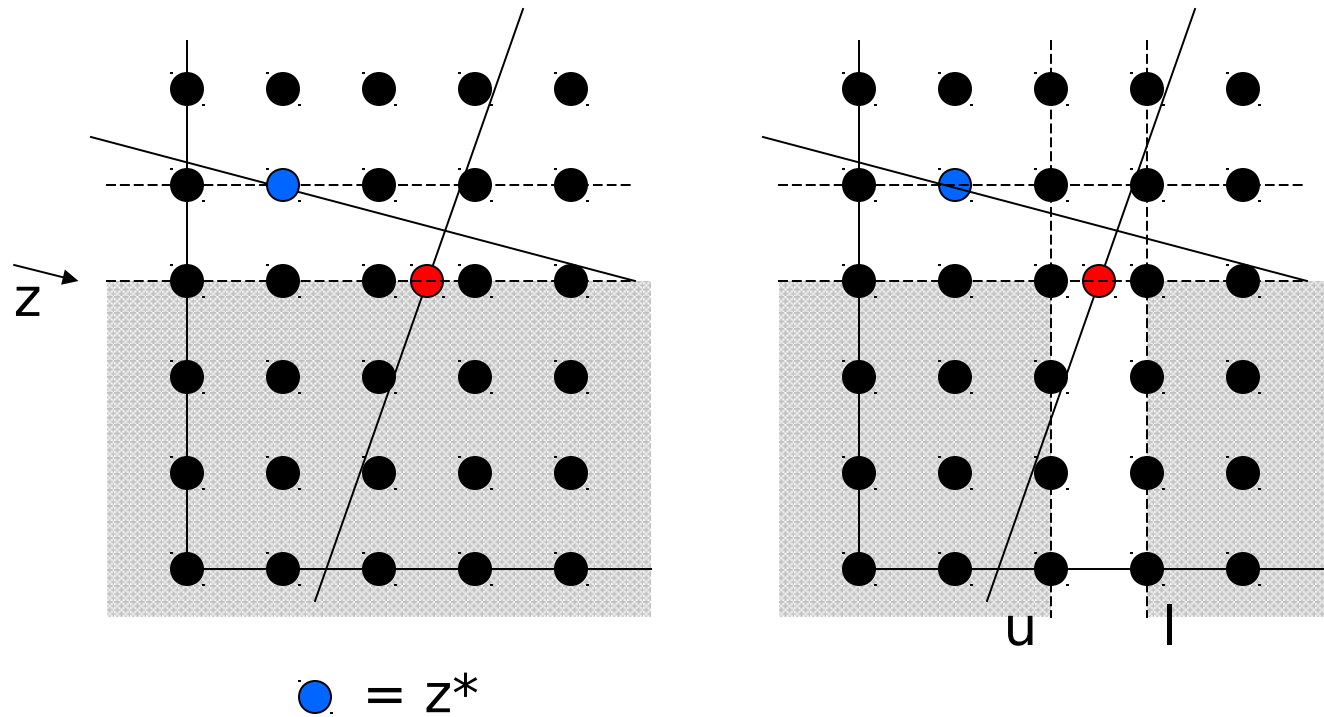
# Example



Find optimal solution.

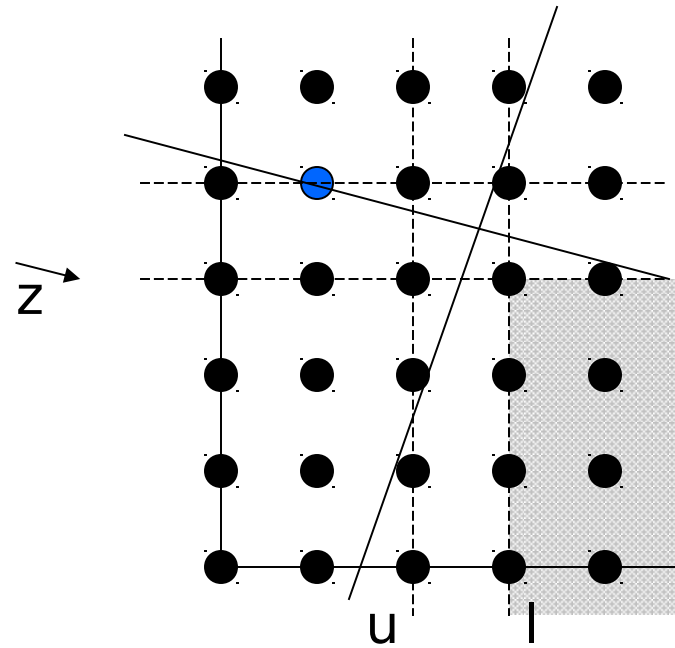
Solution is integral, so return it as current best  $z^*$

# Example



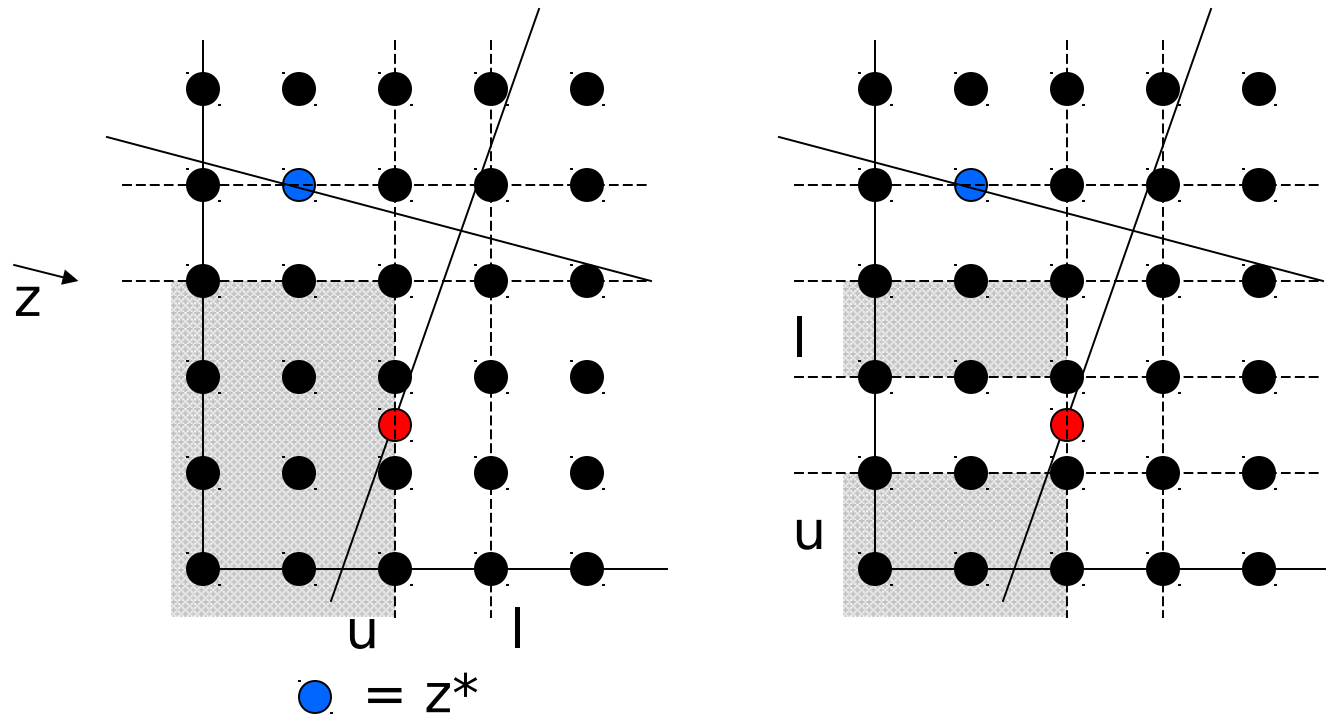
Find optimal solution. It is better than  $z^*$ .  
Cut along  $x$  axis, and make two recursive calls

# Example



Infeasible, Return.

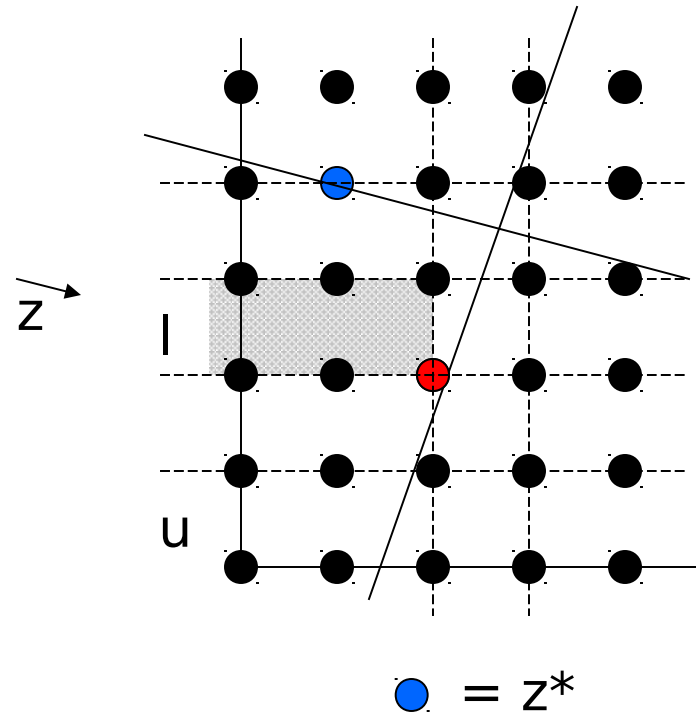
# Example



Find optimal solution. It is better than  $z^*$ .  
Cut along  $y$  axis, and make two recursive calls

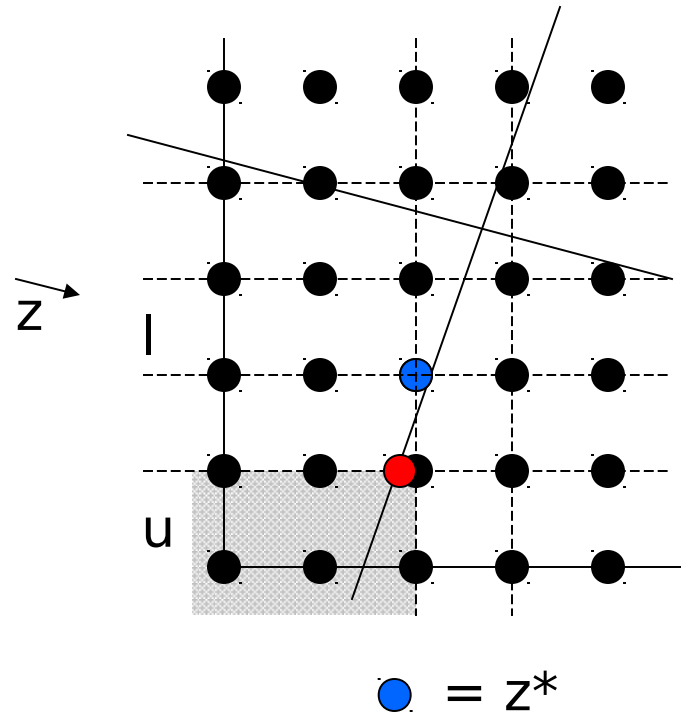


# Example



Find optimal solution. Solution is integral and better than  $z^*$ . Return as new  $z^*$ .

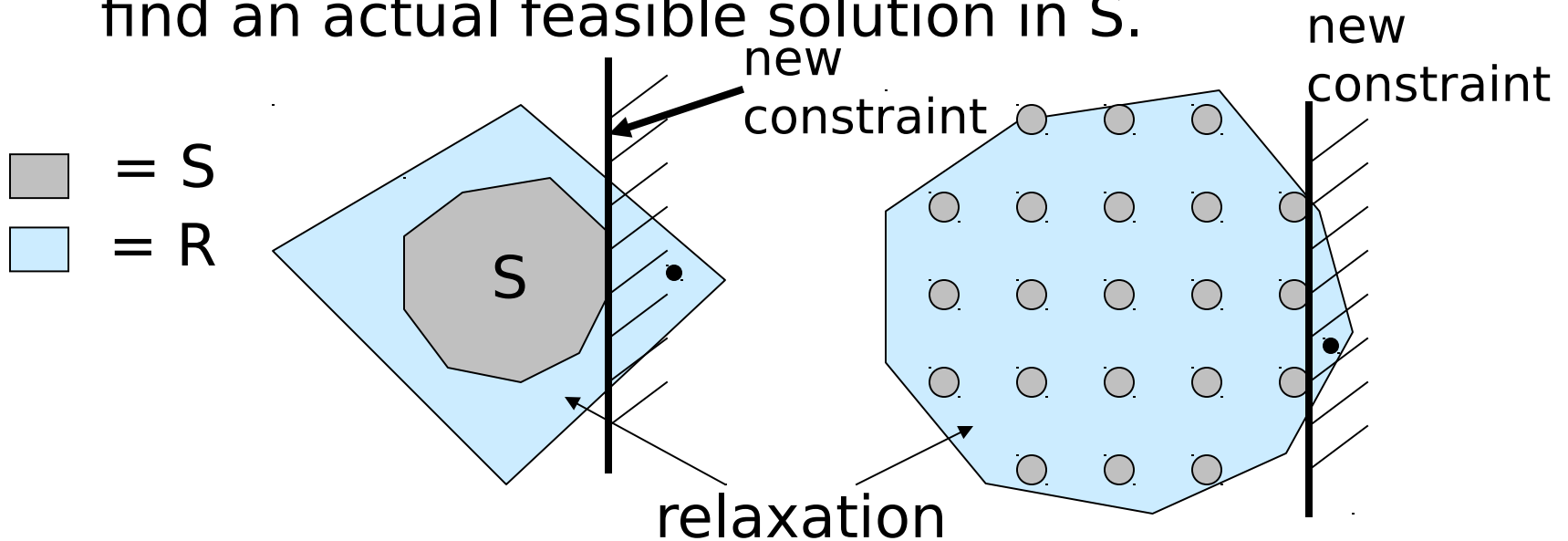
# Example



Find optimal solution. Not as good as  $z^*$ , return.

# Cutting Plane

The idea is to start with a “relaxation”  $R$  of the problem and then add constraints on the fly to find an actual feasible solution in  $S$ .



**Example 1**

**Example 2**

A “linear” relaxation

# Cutting Plane: general algorithm

**minimize:**  $z = c^T x$ , subject to  $x \in S$

**function** CP(R, c)

// R a relaxed set of constraints  $Ax \leq b$   
s.t.  $S \subset \text{polytope}(Ax \leq b)$

**repeat:**

$x = \text{LP}(R, c)$

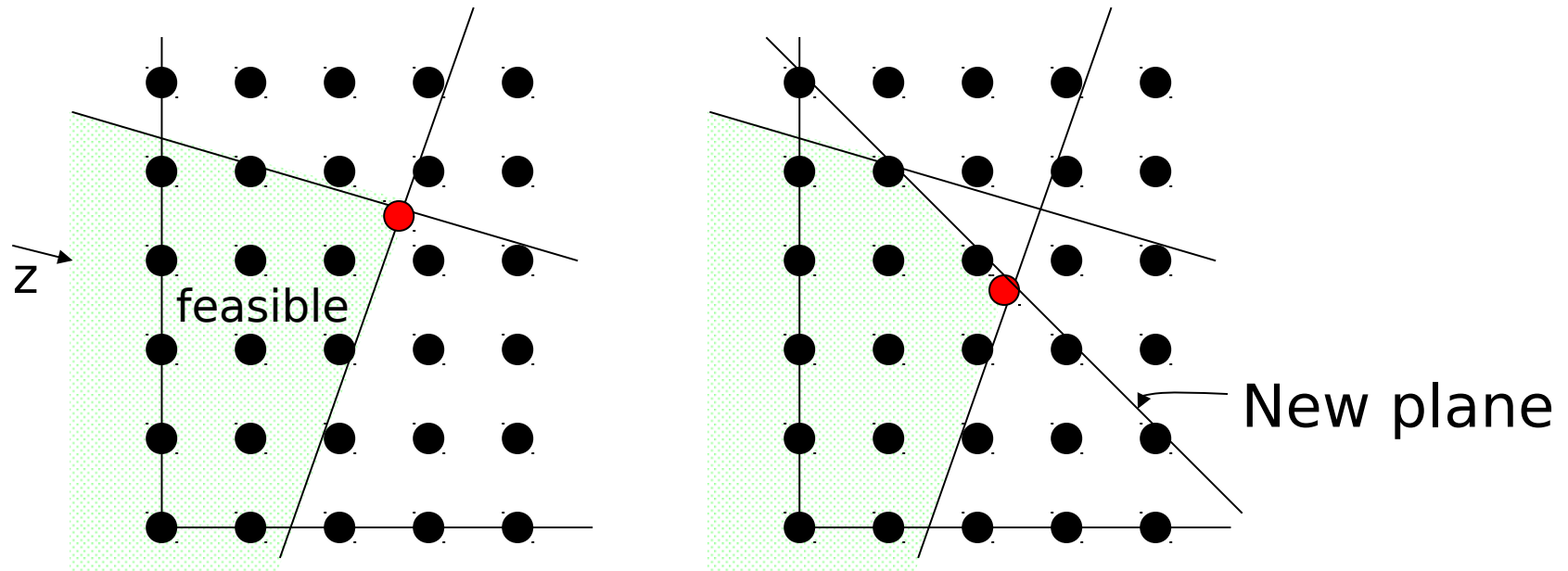
**if**  $x \in S$  **return**  $x$

find an inequality  $r$  satisfied by  $S$ ,  
but violated by  $x$  ( $r$  **separates**  $x$  from  $S$ )

$R = R \cup \{r\}$

Can add multiple inequalities on each iteration

# Cutting Plane



Note that we are removing a corner, and no integer solutions are being excluded.

# Picking the Plane

## **Method 1:** Gomory cuts (1958)

- Cuts are generated from the LP Tableau  
Each row defines a potential cut
- Guaranteed to converge on solution
- General purpose, but inefficient in practice

## **Method 2:** problem specific cuts (templates)

- Consider the problem at hand and generate cuts based on its structure
- A **template** is a problem specific **set** of cuts (probably of exponential size) which  $S$  satisfies. Each round picks a cut from this set.

# Templates for the TSP problem

We consider some example templates used in solutions of the Traveling Salesman Problem.

Recall that  $x_{ij}$  indicates the edge from  $v_i$  to  $v_j$

Assume the symmetric TSP:  $x_{ij} = x_{ji}$

Consider subsets of vertices  $U \subset V$ .

**define:**  $\delta_x(U) = \sum x_{ij}, v_i \in U, v_j \in V-U$

(i.e., the number of times path crosses into/out of  $U$ )

Note: For any  $U$ ,  $\delta_x(U)$  is even.

**Degree Constraints:**  $\delta_x(\{v_i\}) = 2, \quad 1 \leq i \leq n$

**Subtour Constraints:**  $\delta_x(U) \geq 2, \quad U \subset V$  **A**

**template**

There are an exponential number of these

# Templates for the TSP problem

A set of constraints (a template) is **facet-defining** for  $S$  if each constraint is on a facet of the convex hull of  $S$ .

We would like templates which are facet defining since, intuitively, they will more quickly constrain us to the boundary of  $S$ .

The subtour template is facet defining.

In practice the subtour inequalities are not enough to constrain the solution to integral solutions.

Are there other sets of facet defining constraints?



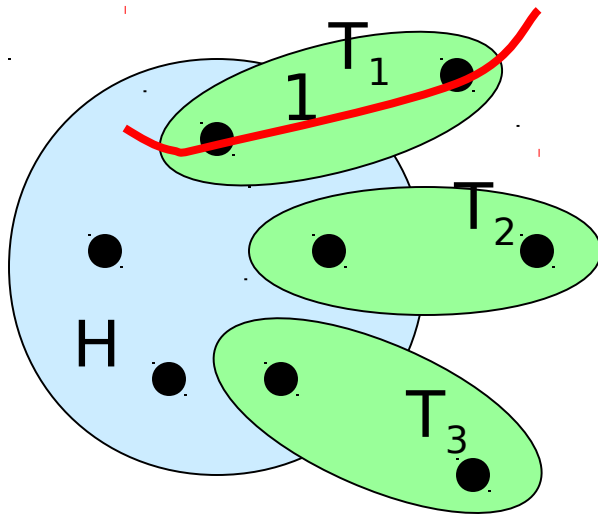
# Templates for the TSP problem

## **Blossom inequalities** (Edmonds 1965):

Defined by  $H$  (handle) and  $T_1, \dots, T_k$  (teeth) satisfying:

$k \geq 3$  and odd,  $|T_i| = 2$

$T_i \cap T_j = \emptyset$ ,  $|H \cap T_i| = 1$ ,  $|T_i \setminus H| = 1$



$$\delta_x(H) + \sum_{i=1}^k \delta_x(T_i) \geq 3k + 1$$

Why? If  $\delta_x(T_i) = 2$ , then  $T_i$  contributes 1 to  $\delta_x(H)$ .

Otherwise,  $\delta_x(T_i) \geq 4$

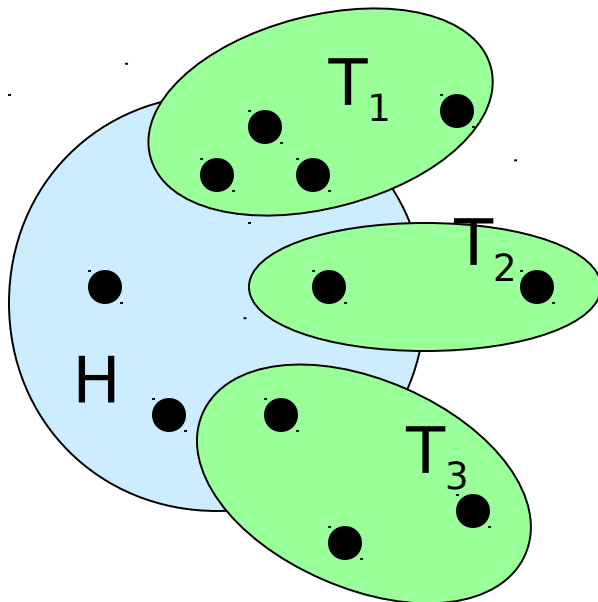
Minimum when  $\delta_x(T_i) = 2$  for all  $i$ . Since  $k$  is odd,  $\delta_x(H) \geq k$  implies

$$\delta_x(H) \geq k + 1$$

# Templates for the TSP problem

## Comb inequalities (Grotschel 1977)

Just generalizes  $T_i$  to be any size. At least one element of each  $T_i$  has to be in and out of  $H$ .



$$\delta_x(H) + \sum_{i=1}^k \delta_x(T_i) \geq 3k + 1$$

# The art of Templates

Picking the right set of templates, and applying them in the right way is the art of solving NP-hard problems with integer programming.

Different problems have different templates.

One needs to find good algorithms for selecting a member of a template that separates  $x$  from  $S$  (can be quite complicated on its own).

Cutting planes often used in conjunction with branch and bound.

Can interleave template cuts with Gomory cuts (e.g., use Gomory cuts when the set of template cuts “dries out”).

# Practical Developments

- **Good formulations**, heuristics and theory  
Goal: to get LP solution as close as possible to IP solution  
Disaggregation, adding constraints (cuts)
- **Preprocessing**  
Automatic methods for reformulation  
Some interesting graph theory is involved
- **Cut generation** (branch-and-cut)  
Add cuts during the branch-and-bound
- **Column generation**  
Improve formulation by introducing an exponential number of variables.