# Operations Research I: Models & Applications
# Integer Programming

Ling-Chieh Kung

Department of Information Management
National Taiwan University

# Road map

- **Introduction to integer programming**.
- Integer programming formulation.
- Facility location problems.
- Machine scheduling problems.
- Vehicle routing problems.

# Integer programming

- We have worked with LPs.
- In some cases, variables must only take **integer values**.
- The subject of formulating and solving models with integer variables is **Integer Programming** (IP).
  - An IP is typically a linear IP (LIP).
  - If the objective function or any functional constraint is nonlinear, it is a nonlinear IP (NLIP).
  - We will focus on linear IP.

# Personnel scheduling (again)

- We know that United Airline developed an LP to determine the number of staffs in each of their service locations.
- The same problem is faced by Taco Bell.
  - It has more than 6500 restaurants in the US.
  - It asks how many staffs to have at each restaurant in each shift.
- Taco Bell developed an Integer Program (i.e., an LP with integer variables) to solve its workforce scheduling problem.
  - The number of staffs is typically **small**! Rounding is very inaccurate.
- \$13 million are saved per year. [1]

[1] Hueter, J., & Swart, W. (1998). An Integrated Labor-Management System for Taco Bell. *Interfaces*, 28(1), 75-91.

# Route selection

- Waste Management Inc. operates an recycling network with 293 landfill sites, 16 waste-to-energy plants, 72 gas-to-energy facilities, 146 recycling plants, 346 transfer stations, and 435 collection depots.
  - 20000 routes must be go through by its vehicles in each day.
- How to determine a route?
  - Construct a network with nodes and edges.
  - Give each edge a **binary** variable: 1 if included and 0 otherwise.
  - Constraints are required to make sure that selected edges are really forming a route.
- A huge IP is constructed to save the company $498 million in operational expenses over a 5-year period.[2]

[2] Sahoo, S., Kim, S., Kim, B., Kraas, B., & Popov, A. (2005). Routing Optimization for Waste Management. *Interfaces*, 35(1), 24-36.

# Road map

- Introduction to integer programming.
- **Integer programming formulation**.
- Facility location problems.
- Machine scheduling problems.
- Vehicle routing problems.

# The knapsack problem

- We start our illustration with the classic **knapsack** problem.
- There are four items to select:

| Item | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Value ($) | 16 | 22 | 12 | 8 |
| Weight(kg) | 5 | 7 | 4 | 3 |

- The knapsack capacity is 10 kg.
- We maximize the total value without exceeding the knapsack capacity.
- The complete formulation is

$$
\begin{array}{rcccccccccc}
\max & 16x_1 & + & 22x_2 & + & 12x_3 & + & 8x_4 & & \\
\text{s.t.} & 5x_1 & + & 7x_2 & + & 4x_3 & + & 3x_4 & \leq & 10 \\
& & & x_i \in \{0,1\} \quad \forall i = 1, ..., 4. & & & & & &
\end{array}
$$

# Requirements on selecting variables

- Integer programming allows us to implement some selection rules.
- At least/most some items:
  - Suppose we must select **at least** one item among items 2, 3, and 4:

$$x_2 + x_3 + x_4 \geq 1.$$

  - Suppose we must select **at most** two items among items 1, 3, and 4:

$$x_1 + x_3 + x_4 \leq 2.$$

# Requirements on selecting variables

- Or:
  - Select item 2 or item 3:
$$x_2 + x_3 \geq 1.$$
  - Select item 2; otherwise, items 3 and 4 togehter:
$$2x_2 + x_3 + x_4 \geq 2.$$
- If-else:
  - If item 2 is selected, select item 3:
$$x_2 \leq x_3.$$
  - If item 1 is selected, do not select items 3 and 4:
$$2(1 - x_1) \geq x_3 + x_4.$$

# At least/most some constraints

- Using a similar technique, we may **flexibly** select constraints.
- Suppose satisfying one of the two constraints

$$g_1(x) \leq b_1 \quad \text{and} \quad g_2(x) \leq b_2.$$

  is enough. How to formulate this situation?
- Let's define a binary variable

$$z = \begin{cases} 0 & \text{if } g_1(x) \leq b_1 \text{ is satisfied,} \\ 1 & \text{if } g_2(x) \leq b_2 \text{ is satisfied.} \end{cases}$$

- With $M_i$ being an upper bound of each LHS, the following two constraints implement what we need:

$$\begin{aligned} g_1(x) - b_1 &\leq M_1 z \\ g_2(x) - b_2 &\leq M_2(1 - z). \end{aligned}$$

# At least/most some constraints

- Suppose at least two of the three constraints

$$g_i(x) \leq b_i, \quad i = 1, 2, 3.$$

  must be satisfied. How to play the same trick?
- Let

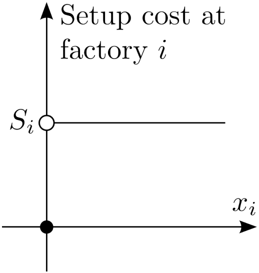
$$z_i = \begin{cases} 1 & \text{if } g_i(x) \leq b_i \text{ is satisfied,} \\ 0 & \text{if } g_i(x) \leq b_i \text{ may be unsatisfied.} \end{cases}$$

- With $M_i$ being an upper bound of each LHS, the following constraints are what we need:

$$\begin{aligned} & g_i(x) - b_i \leq M_i(1 - z_i) \quad \forall i = 1, ..., 3 \\ & z_1 + z_2 + z_3 \geq 2. \end{aligned}$$

# Fixed-charge constraints

- Consider the following example:
- $n$ factories, 1 market, 1 product.
  - $K_i$ is the capacity of factory $i$.
  - $C_i$ is unit production cost at factory $i$.
  - $D$ is the demand of the product.
  - We want to satisfy the demand with the minimum cost.
- **Setup cost** at factory $i$: $S_i$.
  - One needs to pay the setup cost as long as any **positive** amount of products is produced.



# Basic formulation

- Let the decision variables be

$$x_i = \text{production quantity at factory } i,\ i = 1, ..., n,$$

$$y_i = \begin{cases} 1 & \text{if some products are produced at factory } i,\ i = 1, ..., n, \\ 0 & \text{o/w.} \end{cases}$$

- Objective function:

$$\min \sum_{i=1}^{n} C_i x_i + \sum_{i=1}^{n} S_i y_i.$$

- Capacity limitation:

$$x_i \leq K_i \quad \forall i = 1, ..., n.$$

- Demand fulfillment:

$$\sum_{i=1}^{n} x_i \geq D.$$

## Setup costs

- How may we know whether we need to pay the setup cost at factory $i$?
  - If $x_i > 0$, $y_i$ must be 1; if $x_i = 0$, $y_i$ should be 0.
- So the relationship between $x_i$ and $y_i$ should be:

$$x_i \leq K_i y_i \quad \forall i = 1, ..., n.$$

  - If $x_i > 0$, $y_i$ cannot be 0.
  - If $x_i = 0$, $y_i$ can be 0 or 1. Why $y_i$ will always be 0 when $x_i = 0$?
- Finally, binary and nonnegative constraints:

$$x_i \geq 0, y_i \in \{0, 1\} \quad \forall i = 1, ..., n.$$

# Fixed-charge constraints

- The constraint $x_i \leq K_i y_i$ is known as a **fixed-charge constraint**.
- In general, a fixed-charge constraint is

$$x \leq My.$$

  - Both $x$ and $y$ are decision variables.
  - $y \in \{0, 1\}$ is determined by $x$.
  - $M$ must be set to be an **upper bound** of $x$.
  - When $x$ is binary, $x \leq y$ is sufficient.
- We need to make $M$ an upper bound of $x$.
  - For example, $K_i$ is an upper bound of $x_i$ in the factory example. Why?
  - What if there is no capacity limitation?

# Road map

- Introduction to integer programming.
- Integer programming formulation.
- **Facility location problems**.
- Machine scheduling problems.
- Vehicle routing problems.

# Facility location problems

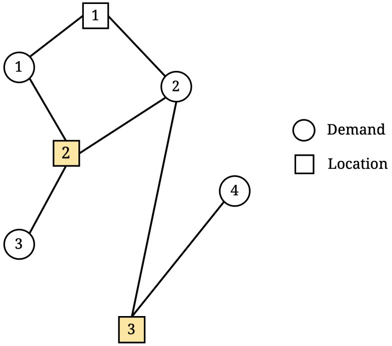
- One typical managerial decision is "where to build my facilities?"
  - Where to open convenience stores?
  - Where to build warehouses or distribution centers?
  - Where to build factories?
  - Where to build power stations, fire stations, or police stations?
- A similar question is "where to locate a scarce resource?"
  - Where to put a limited number of fire engines or ambulances?
  - Where to put a limited number of police officers?
  - Where to put a limited number of ice cream machines?
- These problems are **facility location problems**.
  - In this lecture, we focus on **discrete** facility location problems: We choose a subset of locations from a set of finite locations.

# Facility location problems

- In general, there are some **demand nodes** and some **potential locations**.
  - We build facilities at locations to serve demands.
  - E.g., build distribution centers to ship to retail stores.
  - E.g., build fire stations to cover cities, towns, and villages.
- Facility location problems are typically categorized based on their objective functions.
- In this lecture, we introduce three types of facility location problems:
  - **Set covering problems**: Build a minimum number of facilities to cover all demands.
  - **Maximum covering problems**: Build a given number of facilities to cover as many demands as possible.
  - **Fixed charge location problems**: Finding a balance between benefit of covering demands and cost of building facilities.

# Set covering problems

- Consider a set of demands $I$ and a set of locations $J$.
- The distance (or traveling time) between demand $i$ and location $j$ is $d_{ij} > 0$, $i \in I$, $j \in J$.
- A service level $s > 0$ is given: Demand $i$ is said to be "covered" by location $j$ if $d_{ij} < s$.
- Question: How to allocate as few facilities as possible to cover all demands?

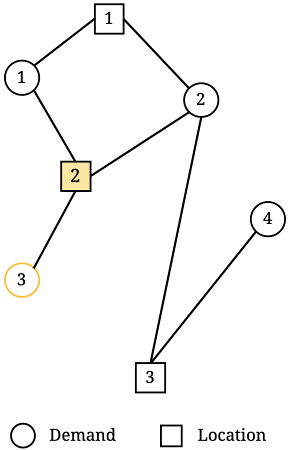


# Set covering problems

- Let's define the following parameter: $a_{ij} = 1$ if $d_{ij} < s$ or 0 otherwise, $i \in I$, $j \in J$.
- Let's define the following variables: $x_j = 1$ if a facility is built at location $j \in J$ or 0 otherwise.
- The complete formulation is

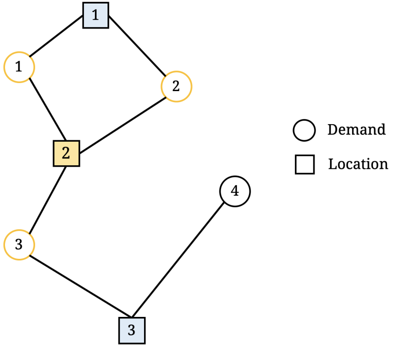
$$
\begin{aligned}
\min \quad & \sum_{j \in J} x_j \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_j \geq 1 \quad \forall i \in I \\
& x_j \in \{0, 1\} \quad \forall j \in J.
\end{aligned}
$$

- The weighted version: $\min \sum_{j \in J} w_j x_j$.



# Maximum covering problems

- Consider a set of demands $I$ and a set of locations $J$.
- The distances $d_{ij}$, service level $s$, and the covering coefficient $a_{ij}$ are also given.
- We are restricted to build at most $p \in \mathbb{N}$ facilities.
- Question: How to allocate at most $p$ facilities to cover as many demands as possible?



# Maximum covering problems

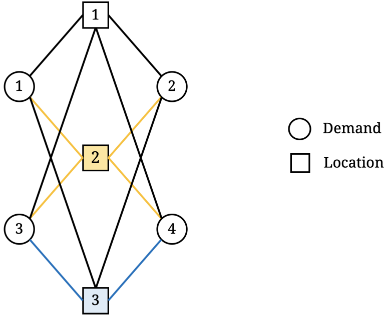
- Still let $x_j = 1$ if a facility is built at location $j \in J$ or 0 otherwise.
- Also let $y_i = 1$ if demand $i \in I$ is covered by any facility or 0 otherwise.
- The complete formulation is

$$
\begin{aligned}
\max \quad & \sum_{i \in I} y_i \\
\text{s.t.} \quad & \sum_{j \in J} a_{ij} x_j \geq y_i \quad \forall i \in I \\
& \sum_{j \in J} x_j \leq p \\
& x_j \in \{0, 1\} \quad \forall j \in J \\
& y_i \in \{0, 1\} \quad \forall i \in I.
\end{aligned}
$$

- The weighted version: $\max \sum_{i \in I} w_i y_i$.

# Fixed charge location problems

- Consider a set of demands $I$ and a set of locations $J$.
- At demand $i$, the demand size is $h_i > 0$.
- The unit shipping cost from location $j$ to demand $i$ is $d_{ij} > 0$.
- The fixed construction cost at location $j$ is $f_j > 0$.
- Question: How to allocate some facilities to minimize the total shipping and construction costs?



# Fixed charge location problems

- We still need $x_j$s: $x_j = 1$ if a facility is built at location $j \in J$ or 0 otherwise.
- We now need $y_{ij}$s: $y_{ij} = 1$ if demand $i \in I$ is served by facility at location $j \in J$ or 0 otherwise.
- The complete formulation is

$$
\begin{aligned}
\min \quad & \sum_{i \in I} \sum_{j \in J} h_i d_{ij} y_{ij} + \sum_{j \in J} f_j x_j \\
\text{s.t.} \quad & y_{ij} \leq x_j \quad \forall i \in I,\ j \in J \\
& \sum_{j \in J} y_{ij} = 1 \quad \forall i \in I \\
& x_j \in \{0, 1\} \quad \forall j \in J \\
& y_i \in \{0, 1\} \quad \forall i \in I.
\end{aligned}
$$

# Fixed charge location problems

- The previous model is the **uncapacitated** version.
  - A facility can serve any amount of demand.
- If facility at location $j$ has a limited capacity $K_j > 0$, we may add the capacity constraint

$$\sum_{i \in I} h_i y_{ij} \leq K_j \quad \forall j \in J.$$

- The **capacitated** version is usually called the capacitated facility location problem (abbreviated as CFL). The uncapacitated one is abbreviated as UFL.

# Remarks

- When to use set covering?
  - When we are required to take care of (almost) everyone.
  - E.g., fire stations and police stations.
- When to use maximum covering?
  - When budgets are limited.
  - E.g., cellular data networks.
- When to use fixed charge location?
  - When service costs depends on distances.
  - E.g., distribution centers.

# Road map

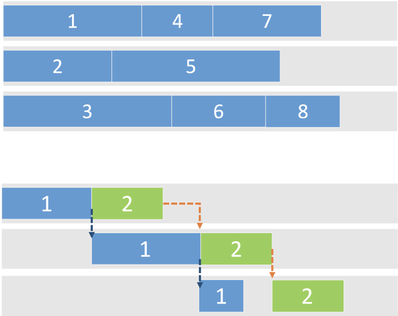
- Introduction to integer programming.
- Integer programming formulation.
- Facility location problems.
- **Machine scheduling problems**.
- Vehicle routing problems.

# Machine scheduling problems

- In many cases, **jobs**/**tasks** must be assigned to **machines**/**agents**.
- As an example, consider a factory producing one product for $n$ customers.
  - Serial production: Only one job can be processed at one time.
  - Each job has its due date.
  - How to schedule the $n$ jobs to minimize the total number of delayed jobs?
- In this example, scheduling is nothing but **sequencing**.
  - Splitting jobs is not helpful.
  - There are $n!$ ways to sequence the $n$ jobs.
  - Is there a way to formulate this problem (so that a solution may be obtained by solving the model)?
- The problems of scheduling jobs/tasks to machines/agents are **machine scheduling problems**.

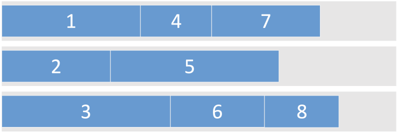
# Machine scheduling problems

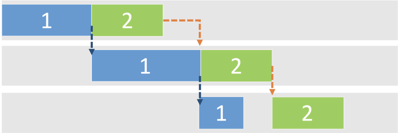
- Machine scheduling problems may be categorized in multiple ways:
- Production mode:
  - Single machine serial production.
  - Multiple parallel machines.
  - Flow shop problems.
  - Job shop problems.



# Machine scheduling problems

- Job splitting:
  - Non-preemptive problems.
  - Preemptive problems.
- Performance measurement:
  - Makespan (the time that all jobs are completed).
  - (Weighted) total completion time.
  - (Weighted) number of delayed jobs.
  - (Weighted) total lateness.
  - (Weighted) total tardiness.
  - And more.



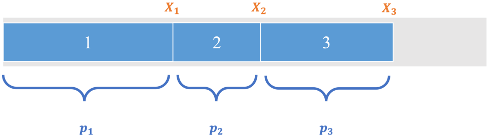
# Minimizing single-machine total completion time

- Consider scheduling $n$ jobs on a single machine.
- Job $j \in J = \{1, 2, ..., n\}$ has **processing time** $p_j$.
- Different schedules give these jobs different **completion times**. The completion time of job $j$ is denoted as $x_j$.
- The machine can process only one job at a time.
- We aim to schedule all the jobs to minimize the total completion time

$$\sum_{j \in J} x_j.$$

# Minimizing single-machine total completion time

- Let's use $x_j$ to be our decision variables.
- Suppose we schedule jobs 1, 2, ..., and $n$ in this order, we will have $x_1 = p_1$, $x_2 = p_1 + p_2$, ..., and $x_n = \sum_{i=1}^{n} p_i$.
- A **Gantt chart** is helpful to illustrate a schedule.




- Obviously, splitting jobs does not help for this problem (Why?).
- Because the machine can start job 2 only after job 1 is completed, we have $x_2 \geq x_1 + p_2$ as a constraint. But what if job 2 should be scheduled before job 1?

# Minimizing single-machine total completion time

- In a feasible schedule, job $i$ is either before or after job $j$, for all $j \neq i$.
- Therefore, we need to satisfy at least one of the following two constraints:
$$x_j \geq x_i + p_j \quad \text{and} \quad x_i \geq x_j + p_i.$$
- Let $z_{ij} = 1$ if job $j$ is before job $i$ or 0 otherwise, $i \in J$, $j \in J$, $i < j$.
- The constraints we need:
$$\begin{aligned} x_i + p_j - x_j &\leq M z_{ij} \\ x_j + p_i - x_i &\leq M(1 - z_{ij}). \end{aligned}$$
- What value of $M$ works?
  - How about $M = \sum_{j \in J} p_j$?

# Minimizing single-machine total completion time

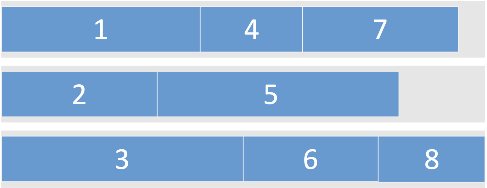
- The complete formulation is

$$
\begin{aligned}
\min \quad & \sum_{j \in J} x_j \\
\text{s.t.} \quad & x_i + p_j - x_j \leq M z_{ij} && \forall i \in J, j \in J, i < j \\
& x_j + p_i - x_i \leq M(1 - z_{ij}) && \forall i \in J, j \in J, i < j \\
& x_j \geq p_j && \forall j \in J \\
& x_j \geq 0 && \forall j \in J \\
& z_{ij} \in \{0, 1\} && \forall i \in J, j \in J, i < j.
\end{aligned}
$$

- While there is a way to optimize the schedule (how?), the problem becomes much harder if a job may be processed only after it is released.
- Let $R_j$ be the **release time** of job $j$. How to add this into the model?

# Minimizing makespan on parallel machines

- Consider scheduling $n$ jobs on $m$ **parallel** machines.
- Job $j \in J = \{1, 2, ..., n\}$ has processing time $p_j$.
- A job can be processed at any machine. However, it can be processed only on one machine.




- Different schedules give these machines different **completion times**.
- The **makespan** of a schedule is the maximum completion time.
- How may we minimize the makespan?

# Minimizing makespan on parallel machines

- As long as some jobs are assigned to a machine, the sequence on that machine does not matter.
- The problem of minimizing makespan is just to **assign** jobs to machines.
- Let $x_{ij} = 1$ if job $j \in J$ is assigned to machine $i \in I$ or 0 otherwise.
- On machine $i \in I$, the last job is completed at

$$\sum_{j \in J} p_j x_{ij}.$$

- The makespan $w$ is the maximum completion time among all machines. We have

$$w \geq \sum_{j \in J} p_j x_{ij} \quad \forall i \in I.$$

# Minimizing makespan on parallel machines

- The complete formulation is

$$\begin{aligned} \min \quad & w \\ \text{s.t.} \quad & w \geq \sum_{j \in J} p_j x_{ij} \quad \forall i \in I \\ & \sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \\ & x_{ij} \in \{0, 1\} \quad \forall i \in I, j \in J. \end{aligned}$$

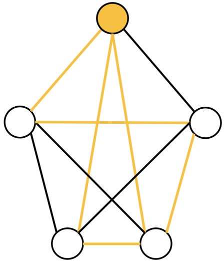
- How may we ensure that $w$ is indeed the makespan?

# Road map

- Introduction to integer programming.
- Integer programming formulation.
- Facility location problems.
- Machine scheduling problems.
- **Vehicle routing problems**.

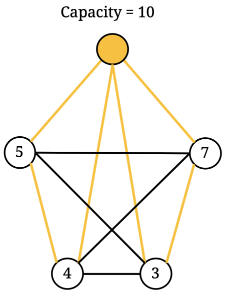
# Vehicle routing problems

- In many cases, we need to deliver/collect items to/from customers in the most efficient way.
- E.g., consider a post officer who needs to deliver to four addresses.
- The shortest path between any pair of two addresses can be obtained.
- This is a **routing** problem: To choose a route starting from the office, passing each address exactly once, and then returning to the office.
- This is a sequencing problem; in total there are $4! = 24$ feasible routes.
- Which route minimizes the total distance (or travel time)?



# Vehicle routing problems

- The problem described above is the famous **traveling salesperson problem**.
  - It assumes that the truck has ample capacity.
- Consider the truck towing bicycles in NTU. It must start at the car pound, pass several locations in NTU, and then return to the origin.
  - However, the truck capacity is quite limited (because too many people violate the parking regulation).
  - The driver needs to find **multiple** routes to cover all the locations.
- The traveling salesperson problem (TSP) is a special case of **vehicle routing problems**.

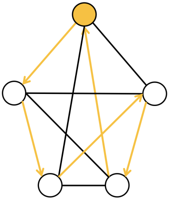


# Traveling salesperson problem

- How to formulate the TSP into an integer program?
- Let's consider a directed complete network $G = (V, E)$.
  - There are $n$ nodes and $n(n-1)$ arcs.
  - The arc weight for arc $(i, j)$ is $d_{ij} > 0$.
- We select a few arcs in $E$ to form a **tour**.
  - To form a tour, we need to select $n$ arcs.
  - These $n$ arcs should form a cycle passing all nodes.
- Let $x_{ij} = 1$ if arc $(i, j) \in E$ is selected or 0 otherwise.
  - The objective:

$$\min \sum_{(i,j)\in E} d_{ij} x_{ij}.$$

  - How to ensure the routing requirement?
  - Is $\sum_{(i,j)\in E} d_{ij} x_{ij} = n$ enough?



# Traveling salesperson problem

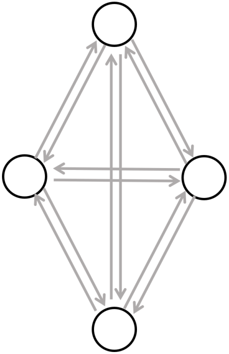
- For node $k \in V$:
  - We must select exactly one incoming arc:

$$\sum_{i \in V, i \neq k} x_{ik} = 1.$$

  - We must select exactly one outgoing arc:

$$\sum_{j \in V, j \neq k} x_{kj} = 1.$$

- Now each node is on a cycle.
- However, these are not enough to prevent **subtours**.

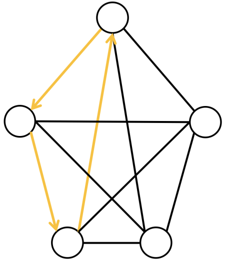


# Eliminating subtours: alternative 1

- There are at least two ways to eliminate subtours.
- For each **subset of nodes** with at least two nodes, we limit the maximum number of arcs selected:

$$\sum_{i \in S, j \in S, i \neq j} x_{ij} \leq |S| - 1 \quad \forall S \subsetneq V, |S| \geq 2.$$

- When we have $n$ nodes, we have $2^n - n - 2$ constraints.
  - $2^n$ ways to choose a subset.
  - $n$ ways to choose a subset of one node.
  - 2 ways to choose a subset of zero node or $n$ nodes.



# Eliminating subtours: alternative 2

- Let $u_i$s represent the order of passing nodes. More precisely, $u_i = k$ if node $i$ is the $k$th node to be passed in a tour.
- We add the following constraints:

$$
\begin{aligned}
& u_1 = 1 & \\
& 2 \leq u_i \leq n & \forall i \in V \setminus \{1\} \\
& u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) & \forall (i,j) \in E, i \neq 1, j \neq 1.
\end{aligned}
$$

  - If $x_{ij} = 0$, there is no constraint for $u_i$ and $u_j$; otherwise, $u_j$ must be larger than $u_i$ by at least 1.
  - If a tour does not contain node 1, the last constraint pushes those $u_i$s to infinity and violates constraint 2.
  - Note that only node 1 is not restricted by these constraints!
- When we have $n$ nodes, we have $n$ additional variables and $2n - 1 + (n-1)(n-2)$ constraints.

# Complete formulation

- The complete formulation is

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} d_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i\in V, i\neq k} x_{ik} = 1 \quad \forall k \in V \\
& \sum_{j\in V, j\neq k} x_{kj} = 1 \quad \forall k \in V \\
& x_{ij} \in \{0,1\} \qquad \forall (i,j) \in E.
\end{aligned}$$

  with either alternative 1 or alternative 2.
- Which alternative is better?