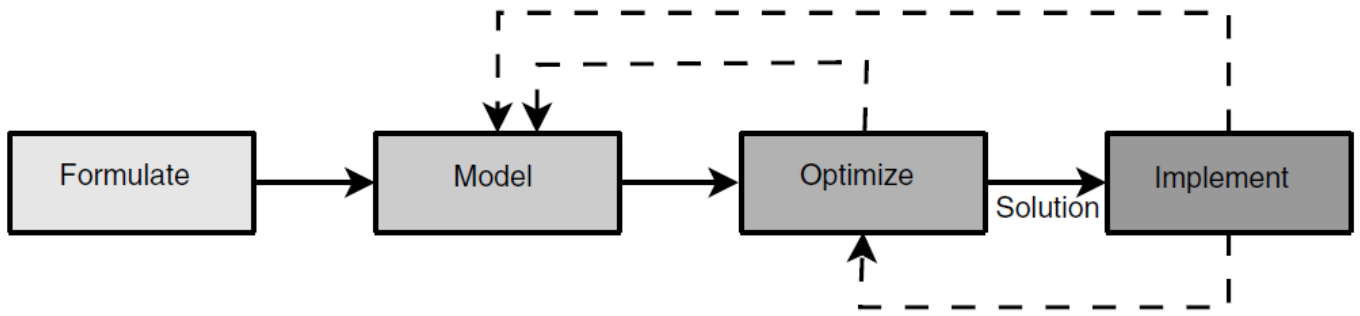
IE552 HEURISTIC METHODS FOR OPTIMIZATION

Lecture Notes 1&2

What is heuristic?

- The word *heuristic* has its origin in the old Greek word *heuriskein*,
  - to find; find out, discover; devise, invent; get, gain, procure
  - the art of discovering new strategies (rules) to solve problems.
- We are trying to **find** solutions to **optimization problems**
  - an optimization problem is the problem of finding the best solution from all feasible solutions.



Optimization Model

- A model could be
  - Linear programming
  - Mixed integer programming
  - Nonlinear programming
  - Constraint Programming

Optimization Model

- In a linear programming optimization problem,
  - both the objective function $c \cdot x$ to be optimized and the constraints $A \cdot x \leq b$ are linear functions.
  - efficient exact algorithms such as the simplex-type method or interior point methods exist
  - In general, there is no reason to use metaheuristics to solve LP continuous problems.
- If a set of variables are discrete then we have a mixed integer model.
- In Nonlinear programming models (NLP) the objective function and/or the constraints are nonlinear.
- Constraint programming (CP) integrates richer modeling tools than the linear expressions. In CP, users declaratively state the constraints on the feasible solutions for a set of decision variables.

A Linear Programming Example

- A company produces two products using two raw materials. The objective is to find the most profitable product mix. Table presents the daily available raw materials, and used amount of raw materials for producst and their profit.

	Product 1	Product 2	Available Amount
Material 1	6	4	24
Material 2	1	2	6
Profit	$5	$4	

$$\text{Max profit} = 5x_1 + 4x_2$$

$$6x_1 + 4x_2 \leq 24$$

$$1x_1 + 2x_2 \leq 6$$

$$x_1, x_2 \geq 0$$

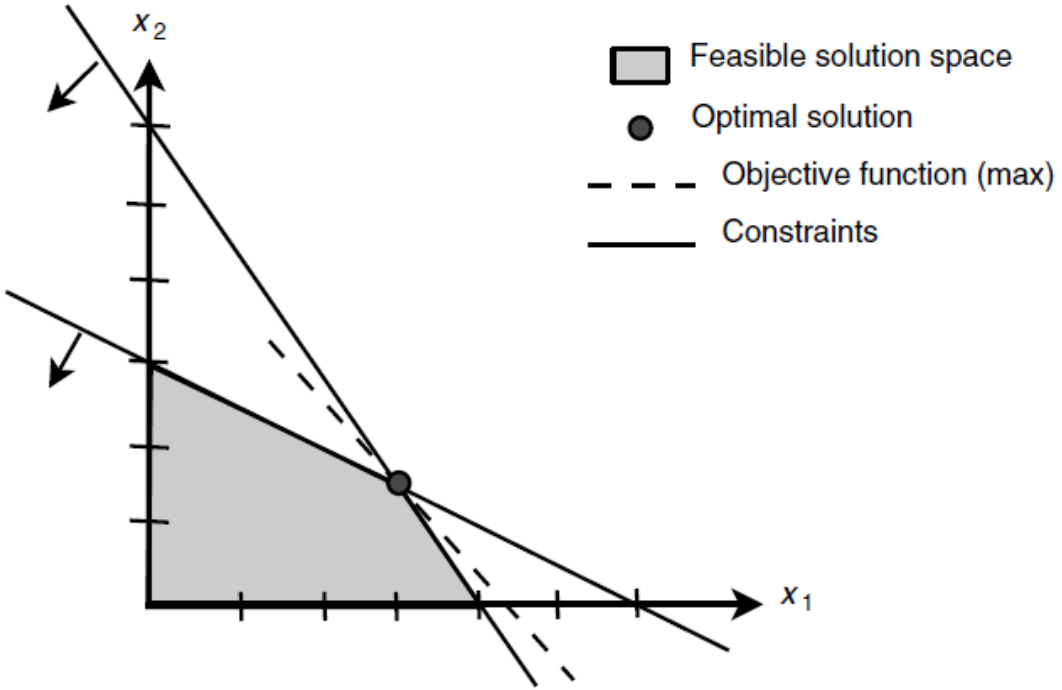
A Linear Programming Example

$$\text{Max profit} = 5x_1 + 4x_2$$

$$6x_1 + 4x_2 \leq 24$$

$$1x_1 + 2x_2 \leq 6$$

$$x_1, x_2 \geq 0$$



Solution: $x_1 = 3\ x_2 = 1.5$

Profit $\$21$

Simplex Algorithm moves one node to other improving the objective function.

An Integer Programming Example (Knapsack)

max $15x_1 + 18x_2 + 16x_3$

subject to $3x_1 + 6x_2 + 4x_3 \leq 9$

$$x_i \in \{0,1\}\ i = 1,2,3,$$

This is an integer programming model. Let's focus on the linear relaxation first in which $0 \leq x_i \leq 1\ i = 1,2,3,$

A greedy heuristic actually solves this relaxation to optimality.

What does greedy mean here?

- Focusing on the best choice at the moment (local optimal) not thinking the future.

An Integer Programming Example (Knapsack)

max $15x_1 + 18x_2 + 16x_3$

subject to $3x_1 + 6x_2 + 4x_3 \leq 9$

$$x_i \in \{0,1\}\ i = 1,2,3,4$$

Greedy heuristic: Calculate the ratio of the objective coefficient to constraint coefficient for each variable. Assign the highest value starting from the highest ratio until the capacity is full.

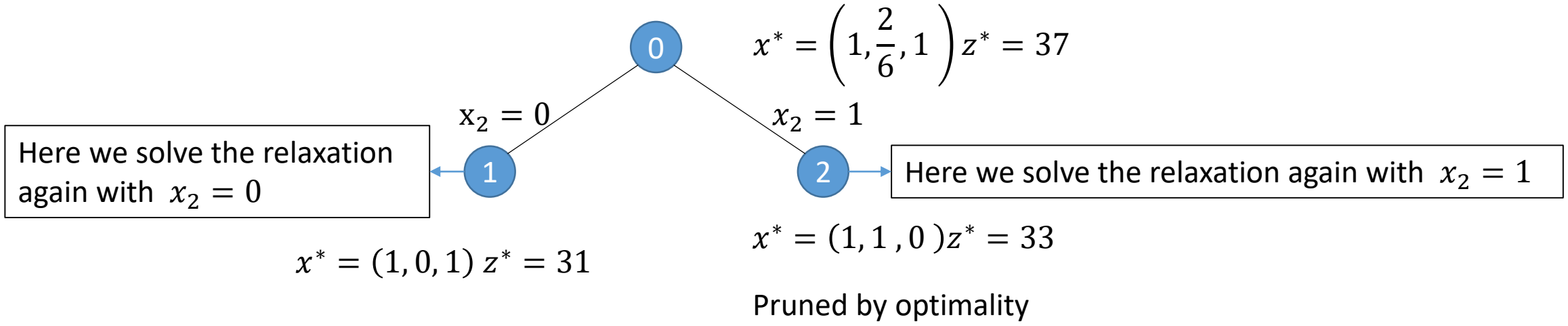
Variable	x_1	x_2	x_3
ratio	$\frac{15}{3} = 5$	$\frac{18}{6} = 3$	$\frac{16}{4} = 4$

The order is x_1, x_3, x_2 So $x_1 = 1, x_3 = 1\ x_2 = \frac{2}{6}$

And the objective function value is $15 + 16 + 18 \times \frac{2}{6} = 37$

An Integer Programming Example (Knapsack)

To find the optimal solution of the IP, we will do branch-and-bound starting from the optimal solution of the relaxation.



Travelling Salesperson Problem

- Perhaps the most popular combinatorial optimization problem is the traveling salesperson problem (TSP).
- It can be formulated as follows: given n cities and a distance matrix d, where each element d_{ij} represents the distance between the cities i and j, find a tour that minimizes the total distance.
- A tour visits each city exactly once (Hamiltonian cycle)
- The size of the search space is n!

Number of Cities n	Size of the Search Space
5	120
10	3, 628, 800
75	2.5×10^{109}

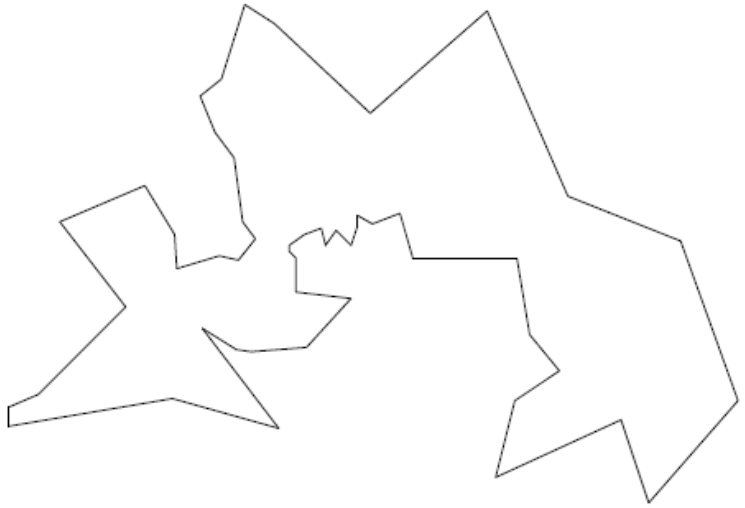


FIGURE 1.4 TSP instance with 52 cities.

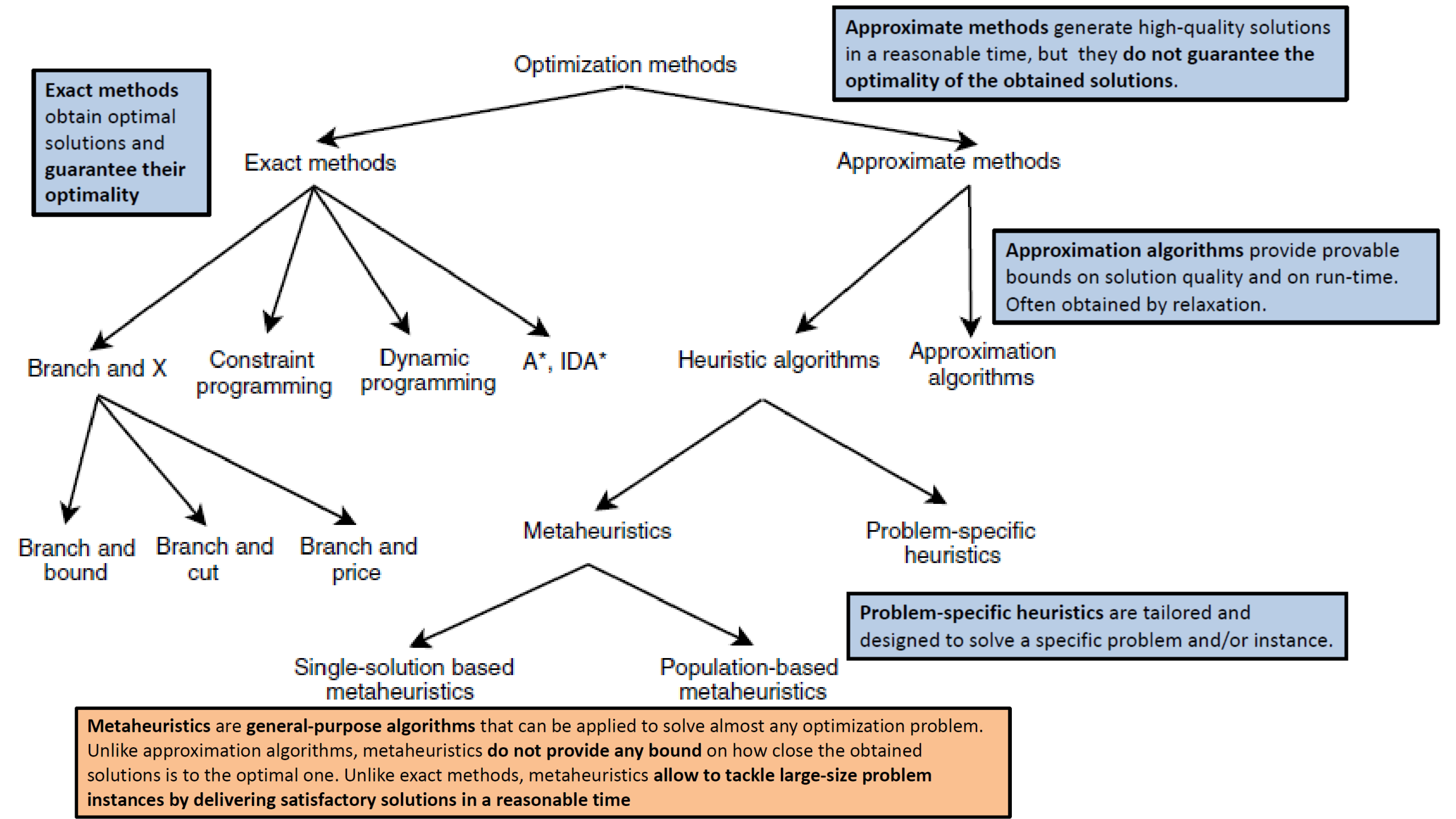
An Integer Linear Programming Formulation of TSP

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

$$\begin{aligned}
x_{ij} &\in \{0,1\} && i,j = 1,\ldots,n; \\
u_i &\in \mathbf{Z} && i = 2,\ldots,n; \\
\sum_{i=1, i\neq j}^{n} x_{ij} &= 1 && j = 1,\ldots,n; \\
\sum_{j=1, j\neq i}^{n} x_{ij} &= 1 && i = 1,\ldots,n; \\
u_i - u_j + (n-1)x_{ij} &\leq n-2 && 2 \leq i \neq j \leq n; \\
1 \leq u_i &\leq n-1 && 2 \leq i \leq n.
\end{aligned}$$

A Greedy Algorithm for TSP

- Start from any node and visit the nearest unvisited node next.
- This is an example of construction heuristic.
- Can you think of an improvement heuristic?



Exact methods obtain optimal solutions and **guarantee their optimality**

Approximate methods generate high-quality solutions in a reasonable time, but they **do not guarantee the optimality of the obtained solutions.**

Approximation algorithms provide provable bounds on solution quality and on run-time. Often obtained by relaxation.

Problem-specific heuristics are tailored and designed to solve a specific problem and/or instance.

Metaheuristics are **general-purpose algorithms** that can be applied to solve almost any optimization problem. Unlike approximation algorithms, metaheuristics **do not provide any bound** on how close the obtained solutions is to the optimal one. Unlike exact methods, metaheuristics **allow to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time**

Complexity Classes

- A complexity class represents the set of all problems that can be solved using a given amount of computational resources. There are two important classes of problems: P and NP

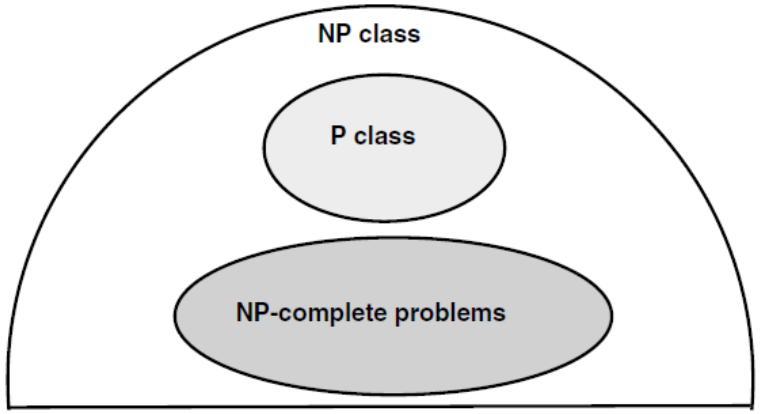


FIGURE 1.6 Complexity classes of decision problems.

- The complexity class P represents the set of all decision problems that can be solved by a deterministic machine in polynomial time.
- A (deterministic) algorithm is polynomial for a decision problem A if its worst complexity is bounded by a polynomial function p(n) where n represents the size of the input instance I.
- The complexity class NP represents the set of all decision problems that can be solved by a nondeterministic algorithm in polynomial time.
- A nondeterministic algorithm contains one or more choice points in which multiple different continuations are possible without any specification of which one will be taken.

Complexity Classes

- A decision problem A ∈ NP is NP-complete if all other problems of class NP are reduced polynomially to the problem
- NP-hard problems are optimization problems whose associated decision problems are NP-complete.
- Most of the real-world optimization problems are NP-hard for which provably efficient algorithms do not exist.
- They require exponential time (unless P =NP) to be solved in optimality.
- Metaheuristics constitute an important alternative to solve this class of problems.

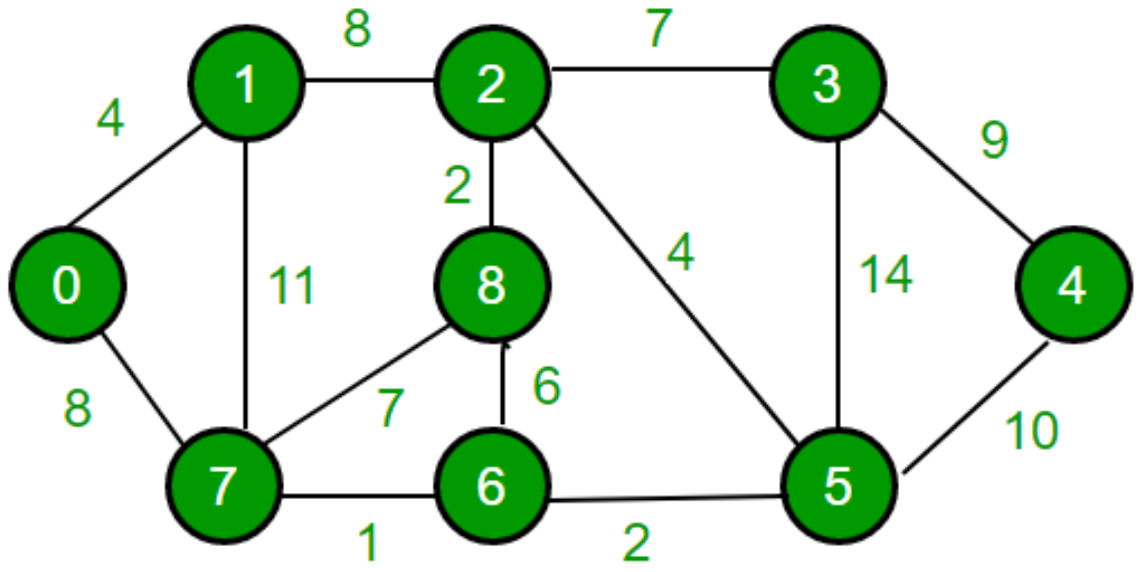
Construction heuristics

- A heuristic algorithm is one that is designed to solve a problem in a faster and more efficient fashion than exact methods by sacrificing optimality for speed.
- Construction heuristic is a type of heuristic method which starts with an empty solution and repeatedly extends the current solution until a complete solution is obtained.
  - They might be called constructive or augmentation heuristics
  - The crucial part is the selection rule
    - It can be random
    - Most common rule is nearest-neighbor strategy. (But what is neighbor?)

Construction heuristics: Shortest Path Example

- Can you think of a heuristic to solve the shortest path problem?

e.g. Shortest path from 0 to 4



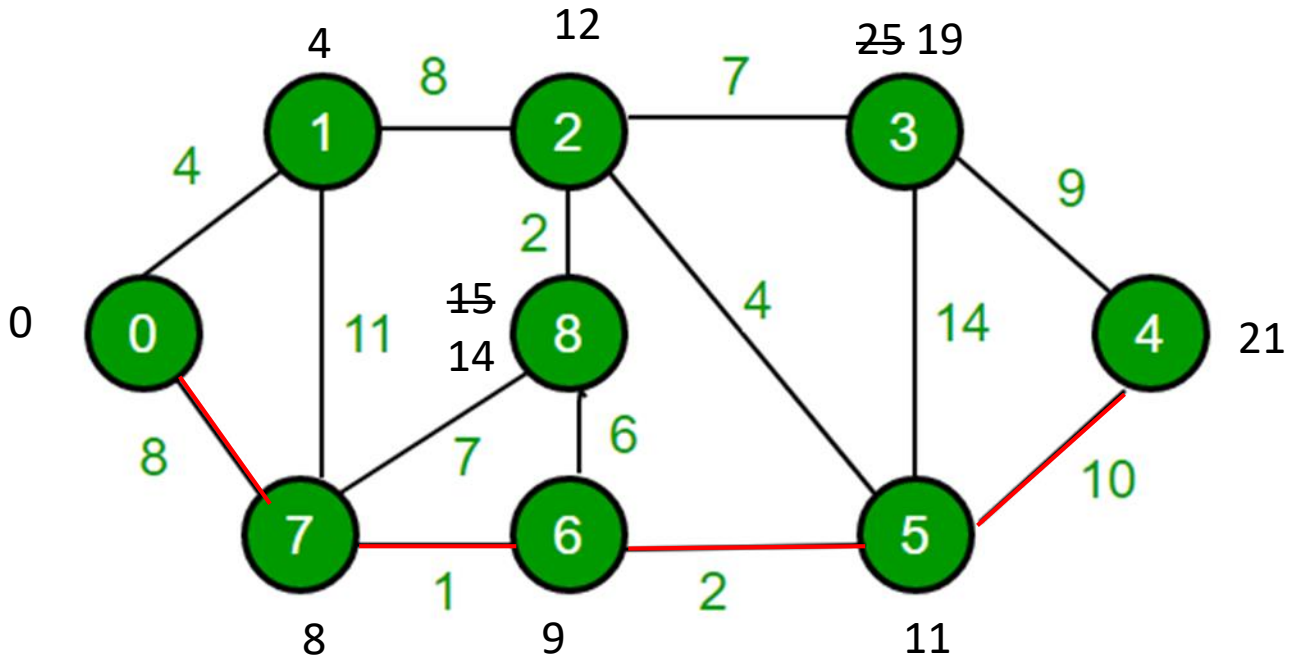
https://levelup.gitconnected.com/finding-the-shortest-path-in-javascript-dijkstras-algorithm-8d16451eea34

Construction heuristics: Shortest Path Example

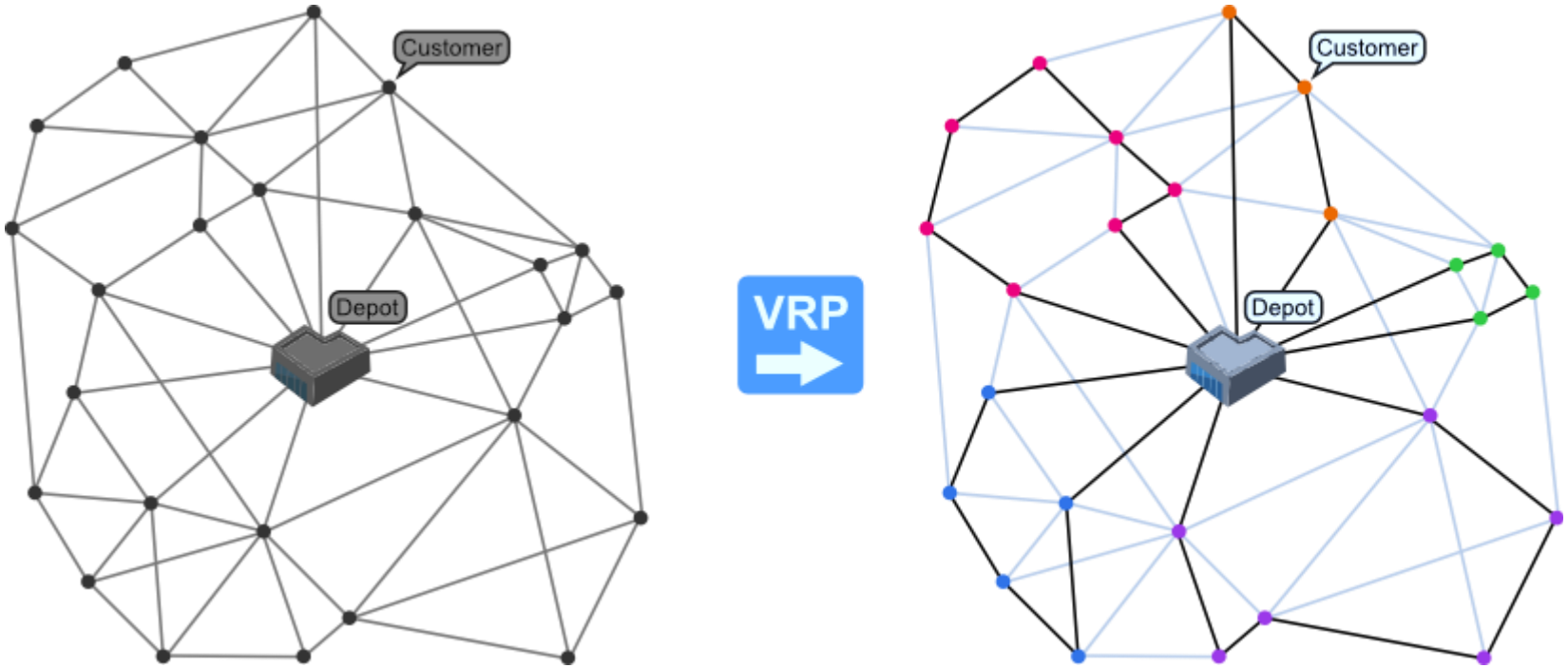
- We actually know how to solve shortest path to optimality!
- Dijkstra's algorithm
  - choose the node with the smallest value as the "current node" and visit all of its neighboring nodes. As we visit each neighbor, we update their tentative distance from the starting node.
  - Once we visit all of the current node's neighbors and update their distances, we mark the current node as "visited." Marking a node as "visited" means that we've arrived at its final cost.
  - We go back to step one and loop until we visits all the nodes in the graph.

Construction heuristics: Shortest Path Example

- We actually know how to solve shortest path to optimality!
- Dijkstra's algorithm



Construction heuristics:
Capacitated Vehicle Routing Example



https://medium.com/cmsa-algorithm-for-the-service-of-the-capacitated/state-of-art-of-the-capacitated-vehicle-rooting-problem-30ad4de6c2e9

Construction heuristics: Capacitated Vehicle Routing Example

- Can you come up with a simple construction heuristic?
  - Remember TSP and nearest-neighbor rule.

Procedure Nearest Neighbor Heuristic

10 Select an arbitrary node j
20 Set $l = j$ and $W = \{1, 2, ..., n\} \setminus \{j\}$
30 While $W \neq \emptyset$ do
40 Let $j \in W$ such that $c_{lj} = \min\{c_{li} \mid j \in W\}$
50 Connect l to j and set $W = W \setminus \{j\}$ and $l = j$.
60 Connect l to the node selected in 10 to form a Hamiltonian cycle.

Construction heuristics: Capacitated Vehicle Routing Example

- Can you come up with a simple construction heuristic?
  - Remember TSP and nearest-neighbor rule.
  - What should we add here?
- Go to the nearest unvisited customer if the capacity allows it! Otherwise return to depot.
- Is it also greedy? Yes!

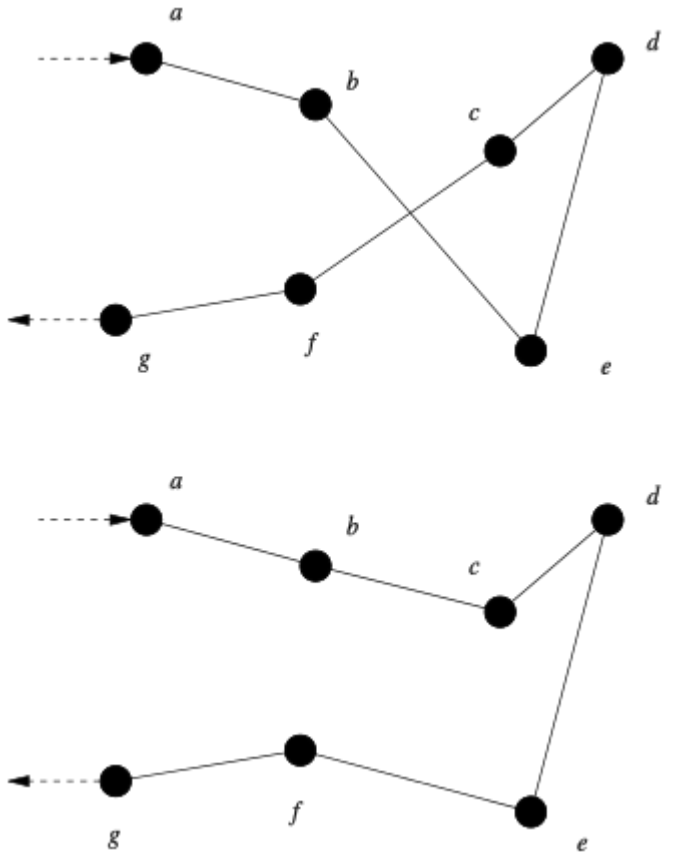
Construction heuristics: Capacitated Vehicle Routing Example

The Clarke and Wright Savings Heuristic

- Exactly one vehicle sent each point
- If we use a single vehicle to serve two points we are going to have savings s(i,j)=d(depot,i)+d(depot,j)- d(i,j)
- Calculate s(i,j) for each pair and order them
- Merge the ones with the largest saving (if feasible)

Improvement Heuristics

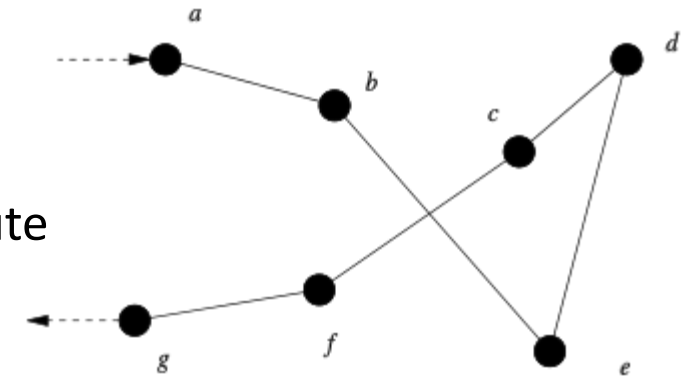
- 2-opt
  - a simple local search algorithm for TSP
  - can be applied to many related problems such as VRP
  - to take a route that crosses over itself and reorder it so that it does not.
  - a complete 2-opt local search will compare every possible valid combination of the swapping mechanism.

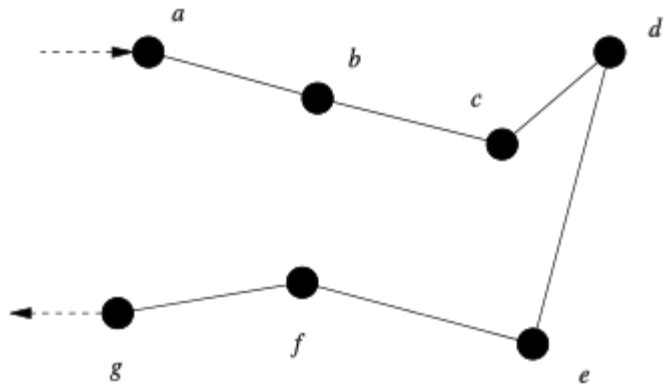


Improvement Heuristics

procedure 2optSwap(route, v1, v2) {
1. take route[0] to route[v1] and add them in order to new_route
2. take route[v1+1] to route[v2] and add them in reverse order to new_route
3. take route[v2+1] to route[end] and add them in order to new_route
return new_route;
}

- Example route: A → B → E → D → C → F → G → H → A
- Example parameters: $v_1 = 1, v_2 = 4$
- Contents of new_route step by step:
  - **(A → B)**
  - A → B → **(C → D → E)**
  - A → B → C → D → E → **(F → G → H → A)**





Improvement Heuristics

- 3-opt for TSP
  - Delete 3 connections (or edges) in a network (or tour) to create 3 sub-tours.
  - Then the 7 different ways of reconnecting the network are analyzed to find the best
  - This process is then repeated for a different set of 3 connections, until all possible combinations have been tried in a network

Set Covering Problem

A city is made up of 5 neighborhoods. We need to decide places of the fire stations to cover whole city. A neighborhood is covered if there is at least one fire station that can reach in at most 5 minutes. The traveling times among neighborhoods and the cost of building a fire station at each neighborhood are given below. Find the least costly solution.

t_{ij}	1	2	3	4	5	f_i
1	0	4	5.5	4.5	6	25
2	4	0	3	6	6	20
3	5.5	5.5	0	6	4	15
4	4.5	6	4	0	7	10
5	6	6	4	7	0	10

We can define a binary parameter a_{ij} equals to 1 if j reaches i in 5 minutes and 0 otherwise. Then we have the constraints in the following form $\sum_j a_{ij} x_j \geq 1 \; \forall i$ where x_j is the binary variable indicating whether we open a station at j or not.

For example the ones that can reach the first neighborhood in time is 1, 2 and 4

Set Covering Problem

A city is made up of 5 neighborhoods. We need to decide places of the fire stations to cover whole city. A neighborhood is covered if there is at least one fire station that can reach in at most 5 minutes. The traveling times among neighborhoods and the cost of building a fire station at each neighborhood are given below. Find the least costly solution.

t_{ij}	1	2	3	4	5	f_i
1	0	4	5.5	4.5	6	25
2	4	0	3	6	6	20
3	5.5	5.5	0	6	4	15
4	4.5	6	4	0	7	10
5	6	6	4	7	0	10

$$\min 25x_1 + 20x_2 + 15x_3 + 10x_4 + 10x_5$$

$$st. \quad x_1 + x_2 + x_4 \geq 1$$
$$x_1 + x_2 \geq 1$$
$$x_2 + x_3 + x_4 + x_5 \geq 1$$
$$x_1 + x_4 \geq 1$$
$$x_3 + x_5 \geq 1$$
$$x_i \in \{0,1\}\ for\ i = 1 \ldots 5$$

Set Covering Problem

- Ideas for a greedy algorithm
  - Since we are minimizing we can focus on cost and open the next costly station until all neighborhoods are covered.
    - The order of locations from cheapest to the most costly would be 4 , 5, 3, 2, 1
    - So we would open, 4, 5, 3, and 2 to cover whole city, and the cost would be 10+10+15+20=55
    - If you look closely, after opening 4 and 5, opening 3 is actually unneccesary.
    - So we can make a better decisions if we check that.
  - We can prioritize the locations which covers more neighborhoods
    - The order of locations would be 1, 2 , 4, 3, 5 (because 1,2,4 covers three neighborhoods, 3 and 5 covers only two.

Set Covering Problem

- We can actually combine these two ideas
- Let d_j be the number of locations j covers. And lets look at the ratio f_j/d_j
- Smaller is better! (less cost more coverage)

1	2	3	4	5
25/3=8.33	20/3=6.67	15/2=7.5	10/3=3.33	10/2=5

- So we open 4 first. It covers 1, 3, 4 (constraints 1,3 and 4 are satisfied)

1	2	3	5
25/1=25	20/1=20	15/1=15	10/1=10

- Next we open 5, delete constraint 5. And finally open 2.

1	2
25/1=25	20/1=20

Set Covering Problem

So the steps of this algorithm is

Step 1: If f_j , = 0, for any j = 1, 2, ..., n, set $x_j = 1$ and remove all covering constraints in which x_j appears.

Step 2: If f_j> 0, for any j = 1, 2, ..., n and x_j does not appear in any of the remaining covering constraints, set $x_j = 0$

Step 3: For each of the remaining variables, determine f_j/d_j where d_j is the number of covering constraints in which x_j appears. Select the variable k for which f_k/d_k is minimum, set $x_j = 1$ and remove all covering constraints in which x_j appears.

Step 4: If there are still constraints, go back to Step 3. Otherwise stop.

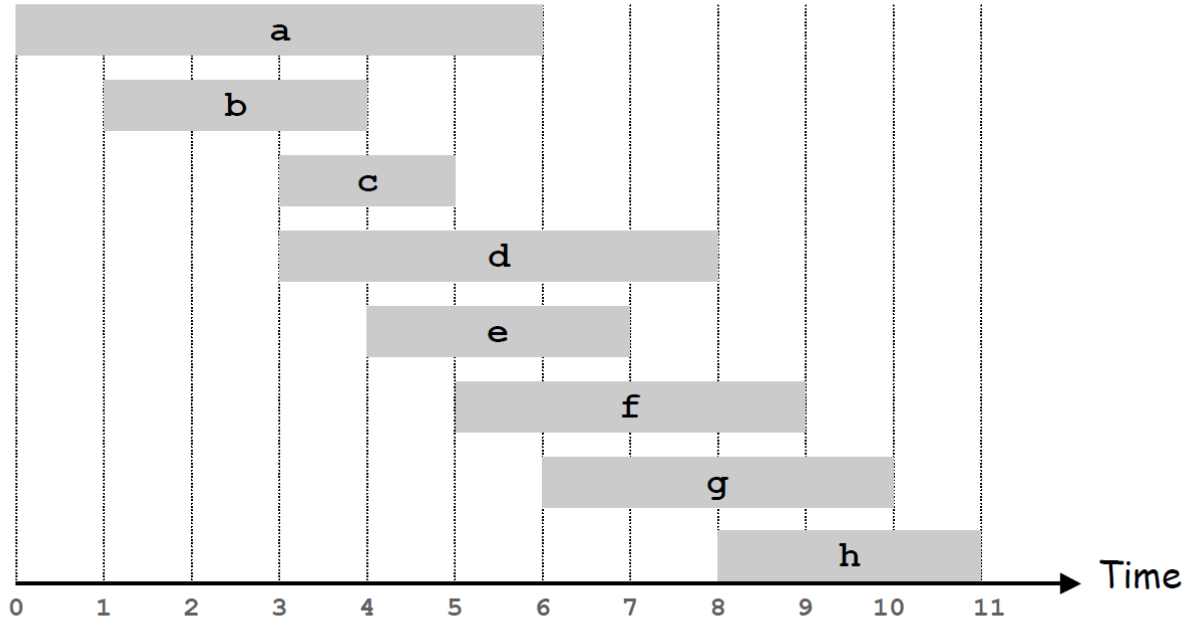
For a bigger example and more information about the problem you can visit https://optimization.cbe.cornell.edu/index.php?title=Set_covering_problem

Set Covering Problem

- With greedy algorithm we now have a feasible solution.
- Can you think of a improvement algorithm?
  - We can close one station and open another one.

Interval Scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy Algorithms

Greedy template: Consider jobs in some natural order. Take each job if it's compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of s_j.
- [Earliest finish time] Consider jobs in ascending order of f_j.
- [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each job j, count the number of conflicting jobs c_j . Schedule in ascending order of c_j.

Which one would you choose?

Interval Scheduling: Greedy Algorithms

counterexample for earliest start time

counterexample for shortest interval

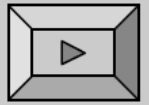
counterexample for fewest conflicts

Interval Scheduling: Greedy Algorithms

- Earliest finish time is the best and it actually gives the optimal solution!

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

                set of jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```



- To see a small example, you can check https://youtu.be/BWlXudP7Unk