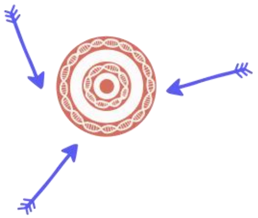
Computational Intelligence

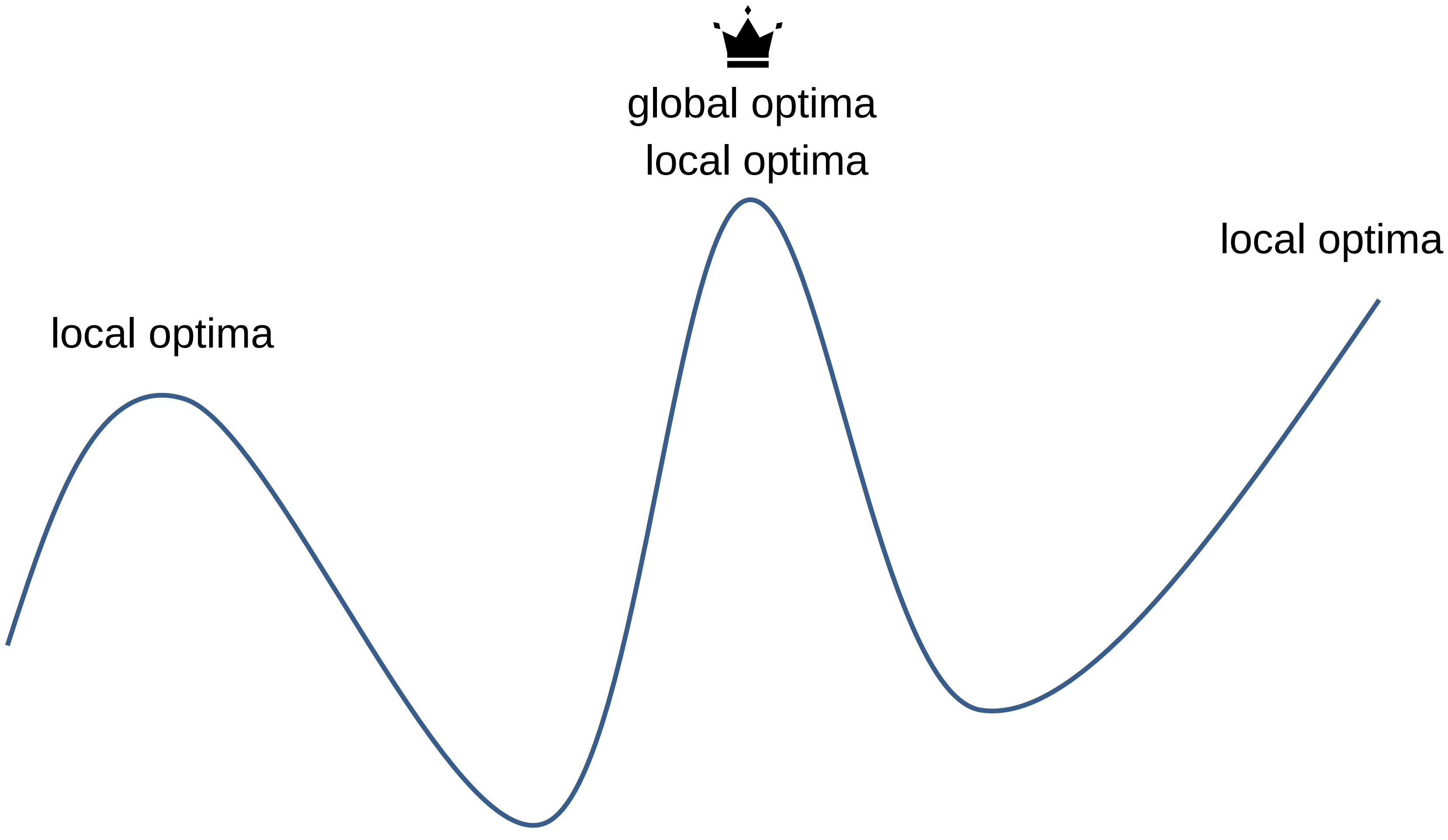
Winter 2024

Characteristics



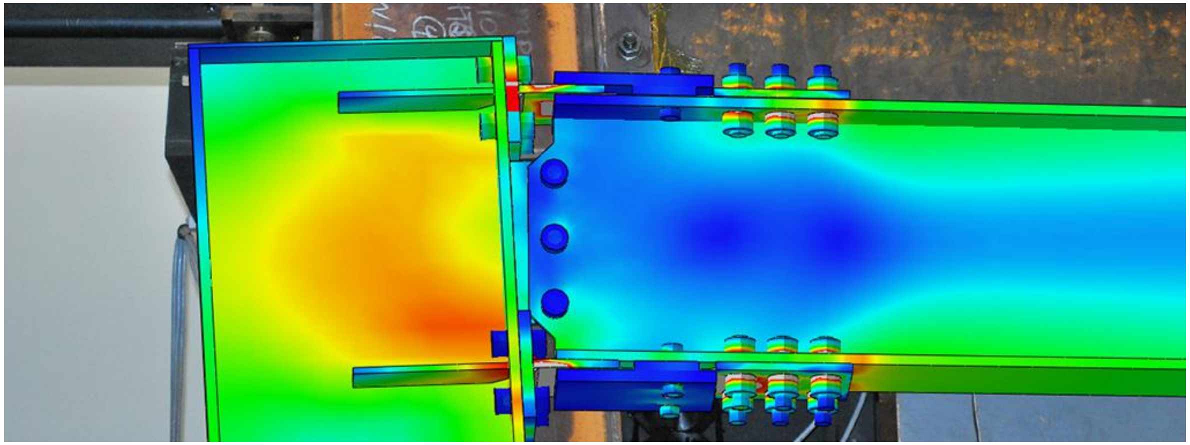
- Meta-heuristics seek good solutions (often near-optimal) at a reasonable computational cost without being able to guarantee optimality, or even to state how close to optimality a particular solution is
- Many references claim that meta-heuristics can avoid solutions being trapped at local optimum
- Meta-heuristics are not a panacea to all problems.

Global versus Local Optima



When to use?

- Meta-heuristics are best and typically used when
  - one has no access to analytical tools, including derivatives (for example if the objective function is non-differentiable)
  - when it does not have an analytical closed form (for example when it is determined as the output of another algorithm: simulation results).



When to use?

- Use meta-heuristics when the problem cannot be solved by classical optimization techniques in polynomial time; especially when the search effort would grow exponentially as the problem size grows

Hybrid with classical techniques

- Even when traditional tools are adequate, metaheuristics may be incorporated with classical optimization tools
  - Integrated genetic algorithm
  - Linear programming embedded genetic algorithm
  - Linear programming driven particle swarm optimization

Attractiveness

- Despite the limits, meta-heuristics attract explosion of interests
- Primarily because of the advancement in computational power and efficiency
  - measured in IPS (Instructions Per Second)

CPU	Year	IPS
Intel 8080	1974	640 KIPS (2 MHz)
Intel 486DX	1992	54 MIPS (66 MHz)
Intel Pentium III	1999	1,354 MIPS (500 MHz)
Intel Core 2 X6800	2006	27,079 MIPS (2.93 GHz)
Intel Core i7 3962X	2011	177,730 MIPS (3.3 GHz)
Intel Core i7 5960X	2014	298,190 MIPS (3.0 GHz)
AMD 3990X	2020	2,356,230 MIPS (4.35 GHz)

Algorithm

- Meta-heuristics are often written in algorithms
- An algorithm is a sequence of steps that takes a set of values (input) and produces a set of values (output)

Algorithm FindLargestNumber

Input: A non-empty list of numbers *L*.

Output: The *largest* number in *L*.

largest ← L_0 **for each** *item* **in** the list $L_{>0}$, **do**

if the *item* > *largest*, **then** *largest* ← the *item*

return *largest*

Requirements of algorithm

- Must take input and generate output
- Must be definite: each step is unambiguous
- It will be terminated after finite steps, with clear termination conditions
- The procedure can be traced by manual calculation

Problem formulation

Minimize $f(X)$

subject to

$$g_i(X) \geq b_i; i = 1,\ldots,n$$

$$h_j(X) = c_j; j = 1,\ldots,m$$

X: vector of decision variables

$f(.)$, $g(.)$, $h(.)$ are general functions

Classes or problems

- Placing restriction on the types of functions and the values that the decision variables can take, we can categorize the problem into the following classes
  - Linear
  - Nonlinear
  - Integer (binary)
  - Mixed integer linear (nonlinear)
  - Combinatorial

Combinatorial optimization

- Decision variables are discrete
- Solution is a set: a sequence of integers
- Examples:
  - Assignment
  - Knapsack
  - Set covering
  - Vehicle routing
  - Traveling salesman

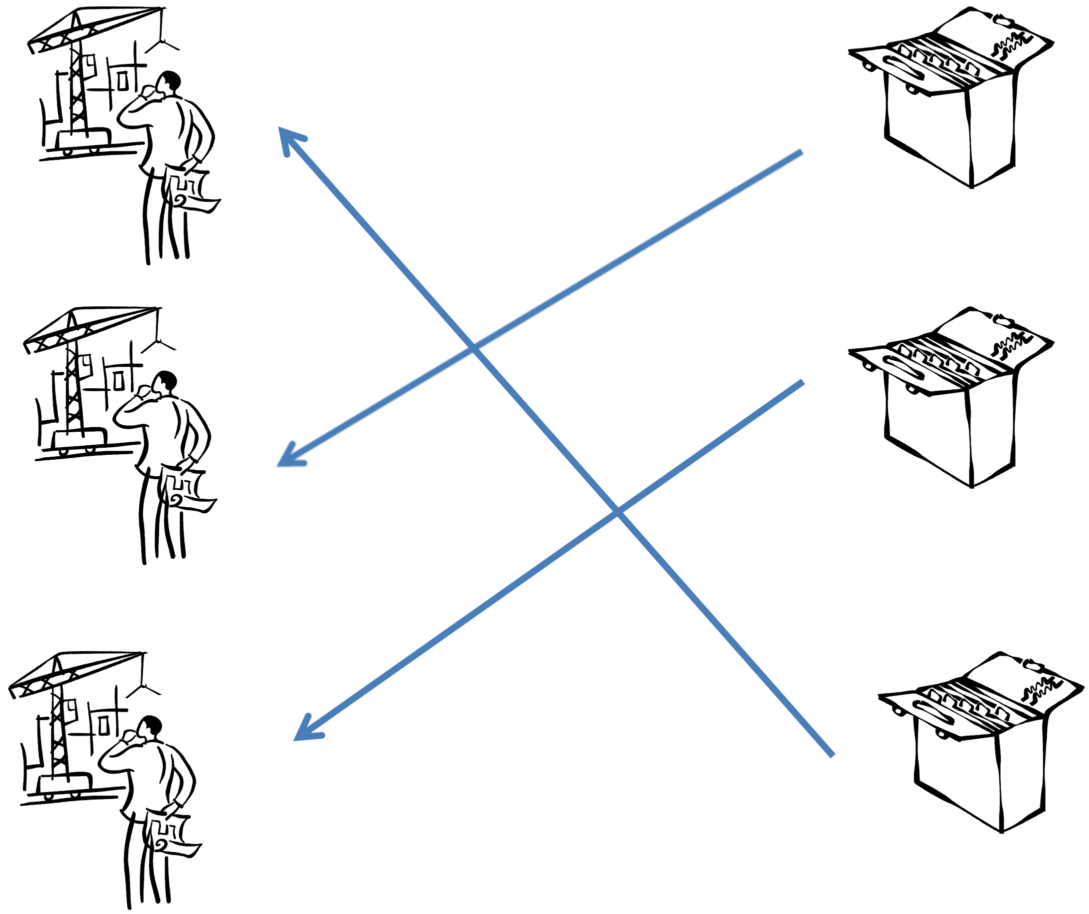
Assignment problem

- A set of n people is available to carry out n tasks. If person i takes task j, it costs c_{ij}. Finding an assignment $\{\pi_1, \pi_2, \ldots, \pi_n\}$ that minimizes $\Sigma c_{i(\pi_i)}$
- Solution is the permutation $\{\pi_1, \pi_2, \ldots, \pi_n\}$ of the number $\{1, 2, \ldots, n\}$
- The solution can also be expressed in a binary matrix

Assignment problem : Illustration

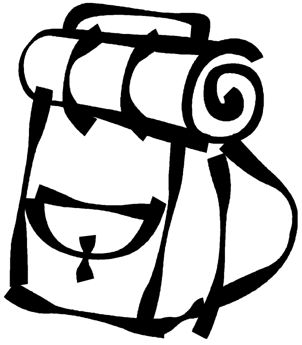
3 1 2

$$\begin{matrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$

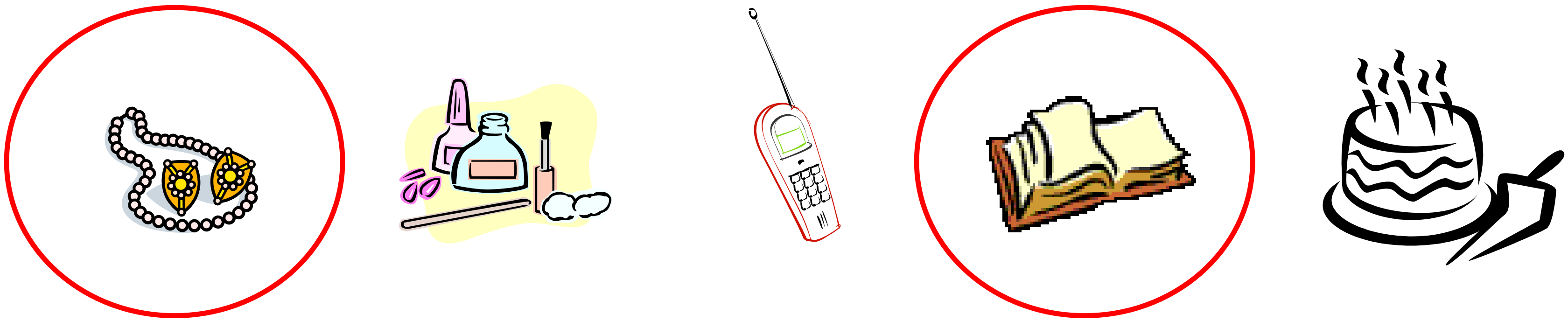


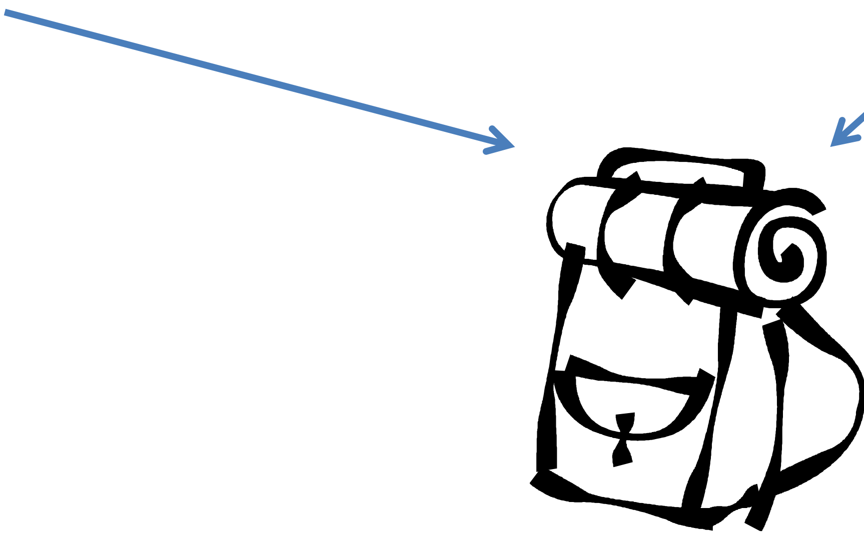
Knapsack problem

- A set of n items is available to be packed into a knapsack with capacity C. Item I has value v_i and uses up c_i units of capacity.
- Determine the subset I of items which should be packed in order to maximize $\sum_{i \in \mathrm{I}} v_i$ such that $\sum_{i \in \mathrm{I}} c_i \leq C$



Knapsack problem : Illustration





1 4

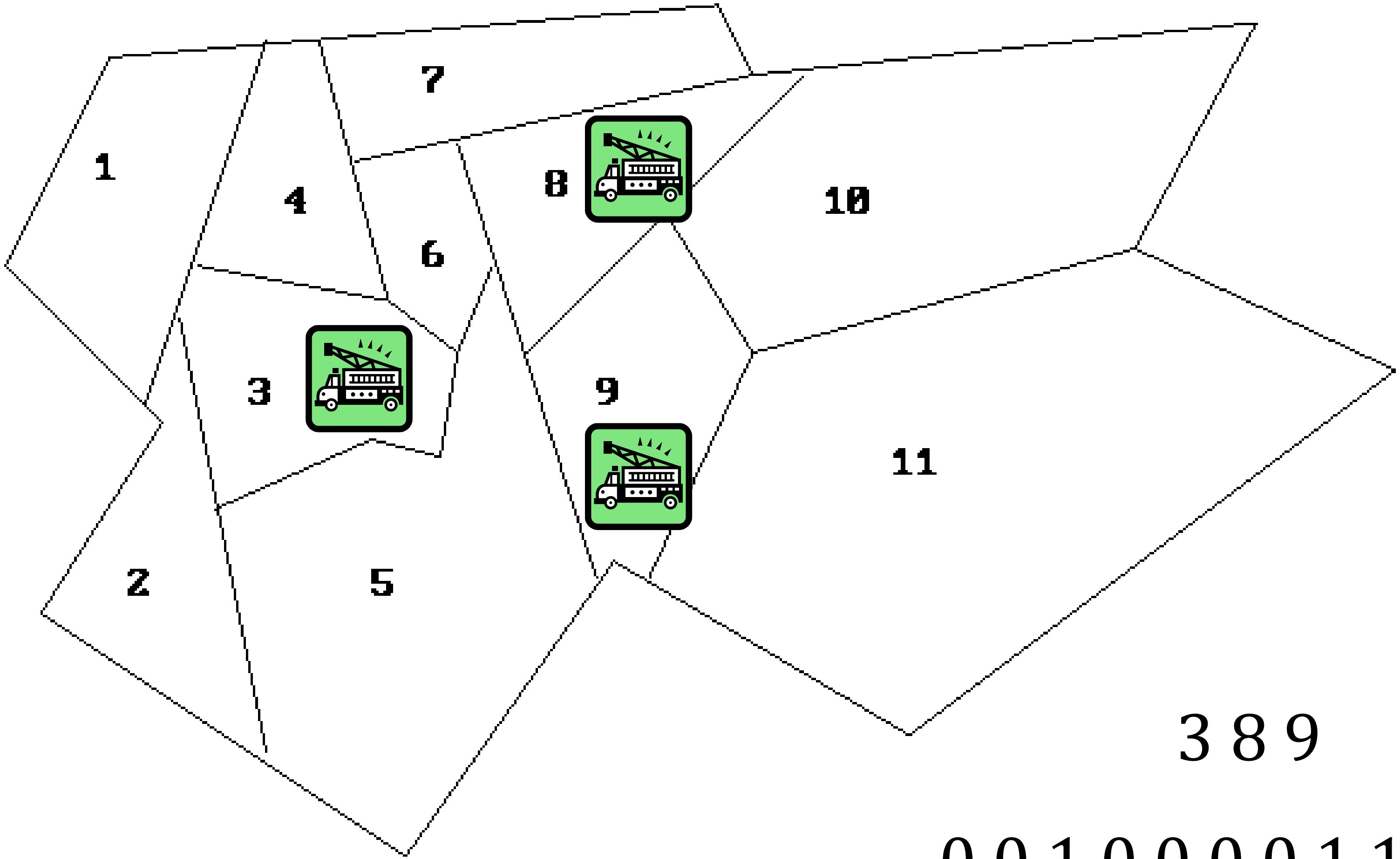
1 0 0 1 0

Set covering problem

- A family of m subsets collectively contains n items such that subset S_i contains n_i ($\leqq n$) items.
- Select k ($<m$) subsets $\{ T_1, T_2, \ldots, T_k \}$ such that $|\bigcup_{j=1}^{k} T_j| = n$

  So as to minimize $\sum_{j=1}^{k} c_j$

Set covering problem: Illustration



3 8 9

0 0 1 0 0 0 0 1 1 0 0

http://mat.gsia.cmu.edu/

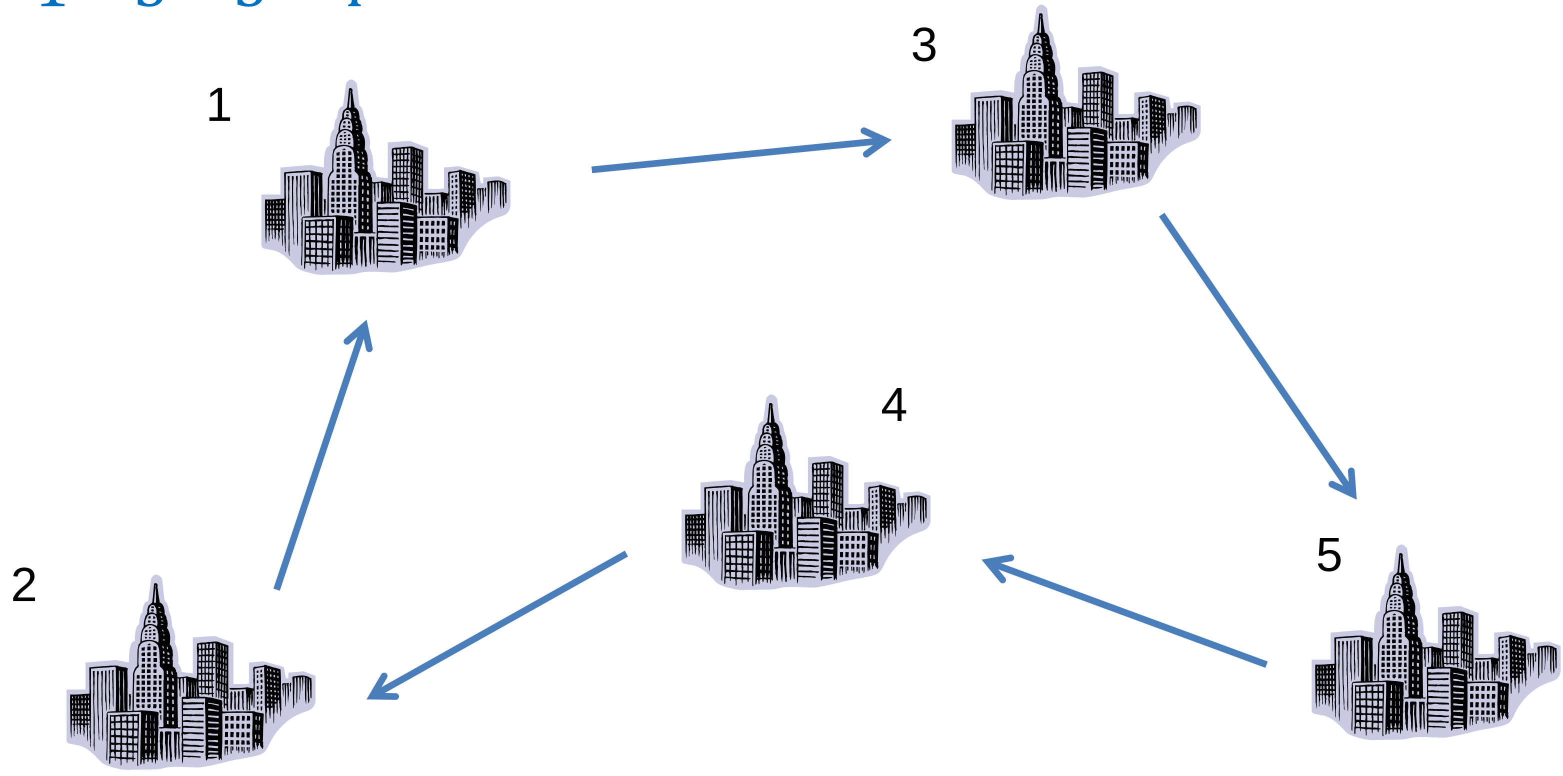
Traveling salesman problem

- Given a finite number of cities along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities exactly once and returning to starting point.
- Simplified form:

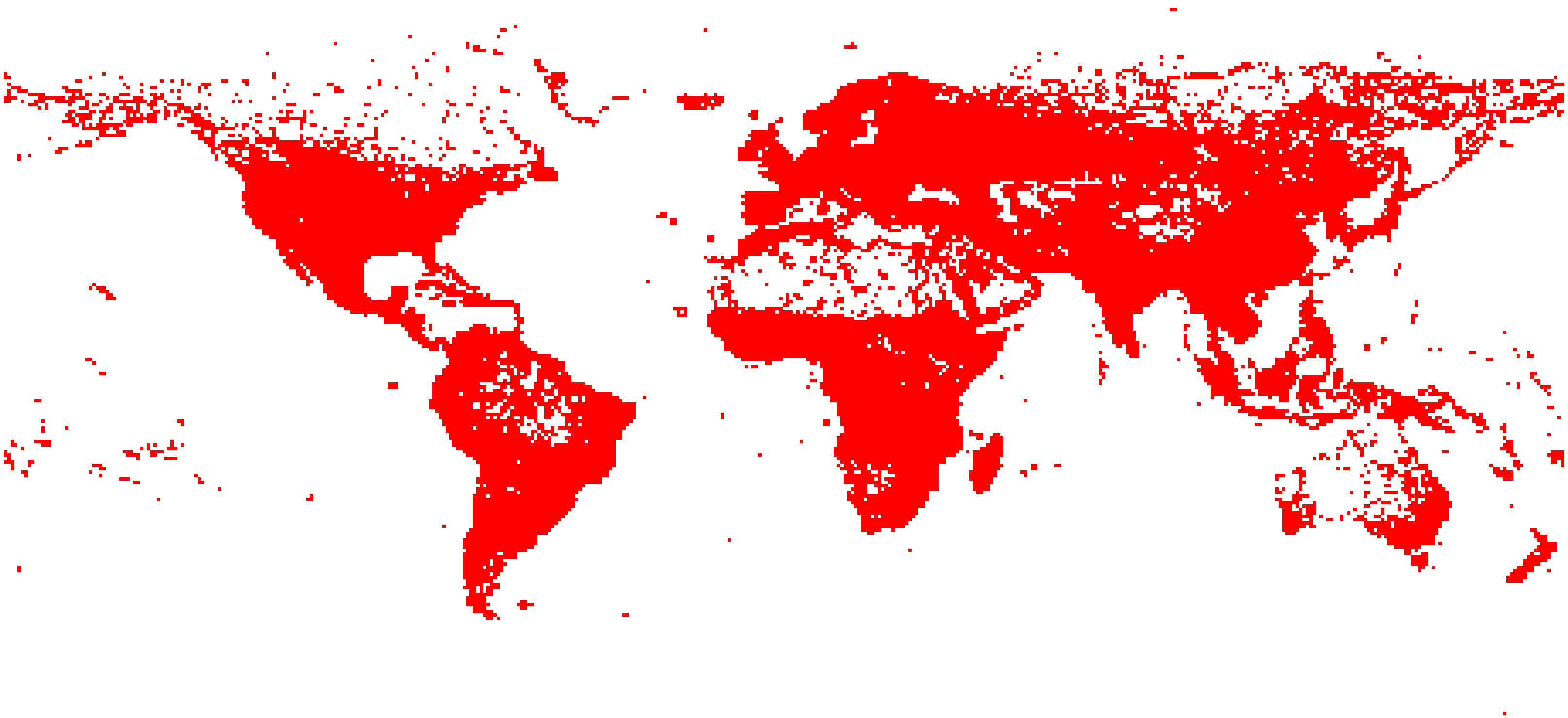
$$
\begin{array}{ll}
\text{Minimize} & \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{Subject to} & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n, \\
& \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n, \\
& x_{ij} = 0 \text{ or } 1 \\
& \text{no sub-tours allowed}
\end{array}
$$

Traveling salesman problem: Illustration

2 1 3 5 4



Travel around the globe



1,904,711 cities http://www.math.uwaterloo.ca/tsp/world/index.html

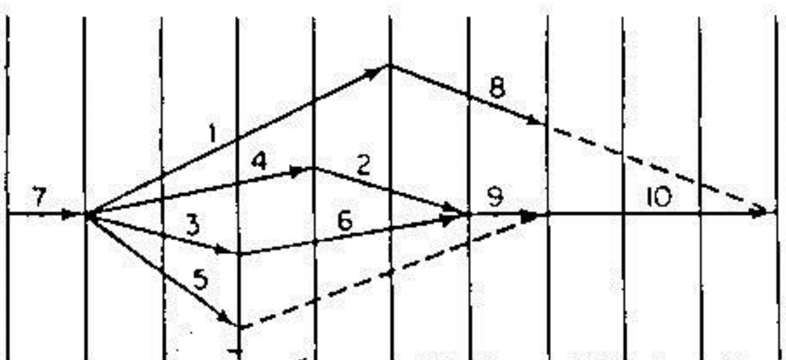
Limited resource allocation

- An optimization problem may be analogous to each other
- A construction project consists of *n* tasks. Due to limited resources, we have to arrange the priority of tasks so as to ensure daily resource requirements would not exceed the limits.
- Priority List of Tasks may be analogous to Sequence of Cities

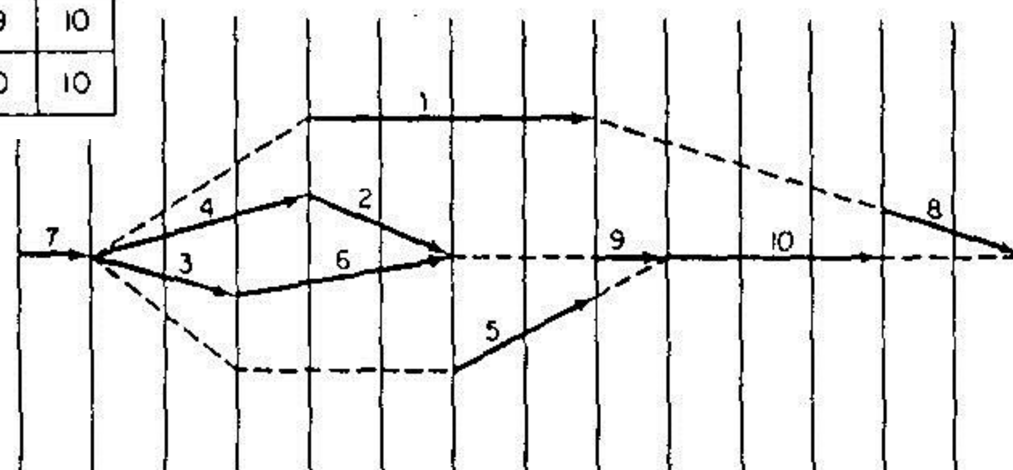
Example of resource allocation

Daily manpower limit =10
Priority list
{2 3 4 6 1 5 7 9 10 8} → 14 days

How about {1 4 5 2 9 7 10 8 6 3}?



Day	1	2	3	4	5	6	7	8	9	10
Manpower requirements	7	13	13	11	9	16	17	10	10	10



Day	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Manpower requirements	7	7	7	10	9	9	6	6	9	10	10	10	8	8

Vehicle routing problem

- A depot has m vehicles available to make deliveries to n customers. The capacity of vehicle k is C_k units, where customer i requires c_i units. The distance between customers i and j is d_{ij}. No vehicle may travel more than D units.
- Allocate customers to vehicle and find the order in which each vehicle visits its customers so as to minimize $\sum_{k=1}^{m}\sum_{i=0}^{n_k} d_{\pi_{i,k},\pi_{i+1,k}}$

Vehicle routing problem (Cont.)

such that $\sum_{i=0}^{n_k} c_{\pi_{i,k}} \le C_k \,;\, k = 1,2,\ldots,m$

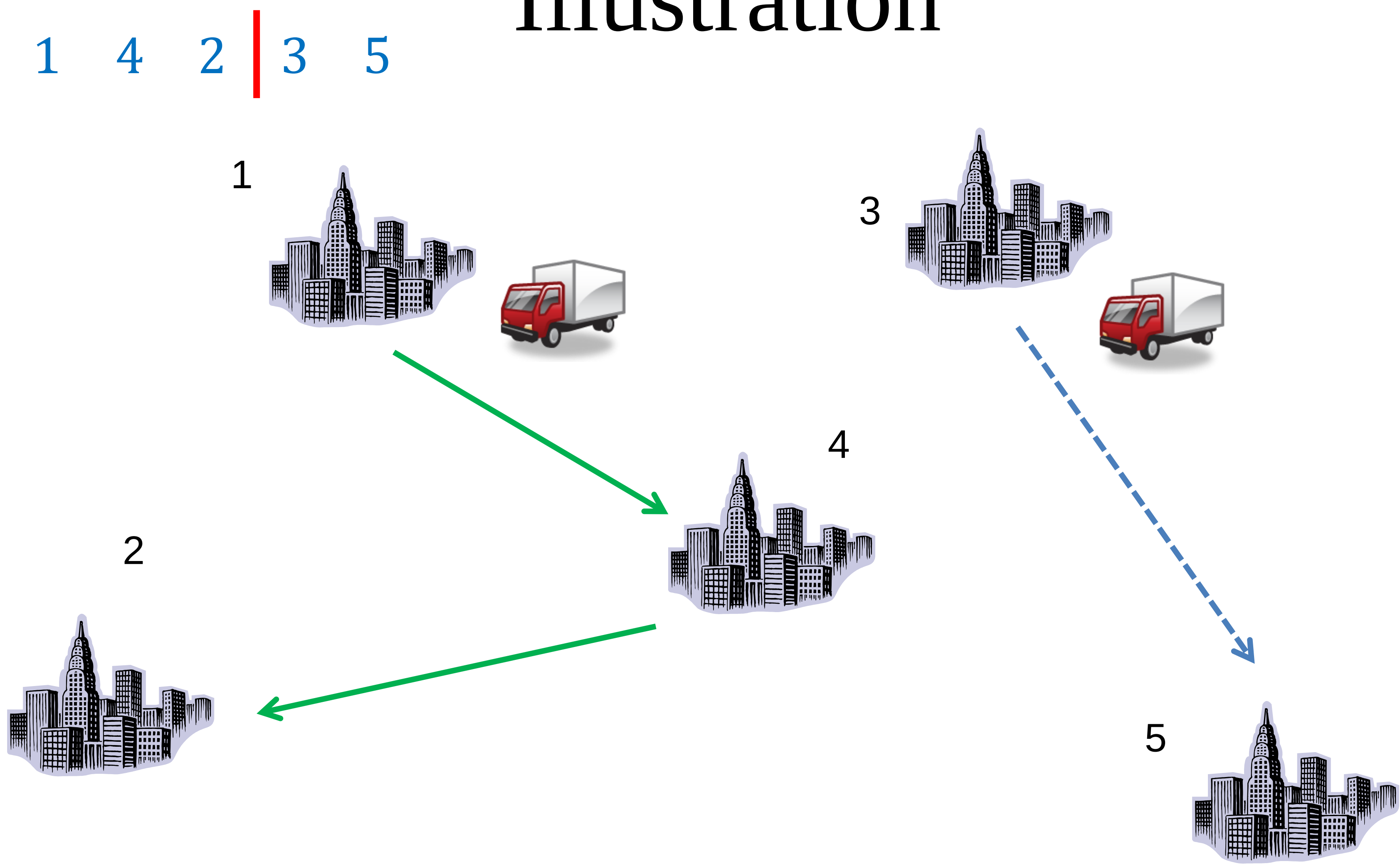
$$\sum_{i=0}^{n_k} d_{\pi_{i,k},\pi_{i+1,k}} \le D \,;\, k = 1,2,\ldots,m$$

$$\sum_{k=1}^{m} n_k = n$$

where vehicle k visits n_k customers.

- The solution is represented by the permutation $\{\underbrace{\pi_{1,1},\ldots,\pi_{n_1,1}}_{\text{Vehicle 1}},\ldots,\underbrace{\pi_{1,m},\ldots,\pi_{n_m,m}}_{\text{Vehicle m}}\}$ of the number $\{1, \ldots, n\}$

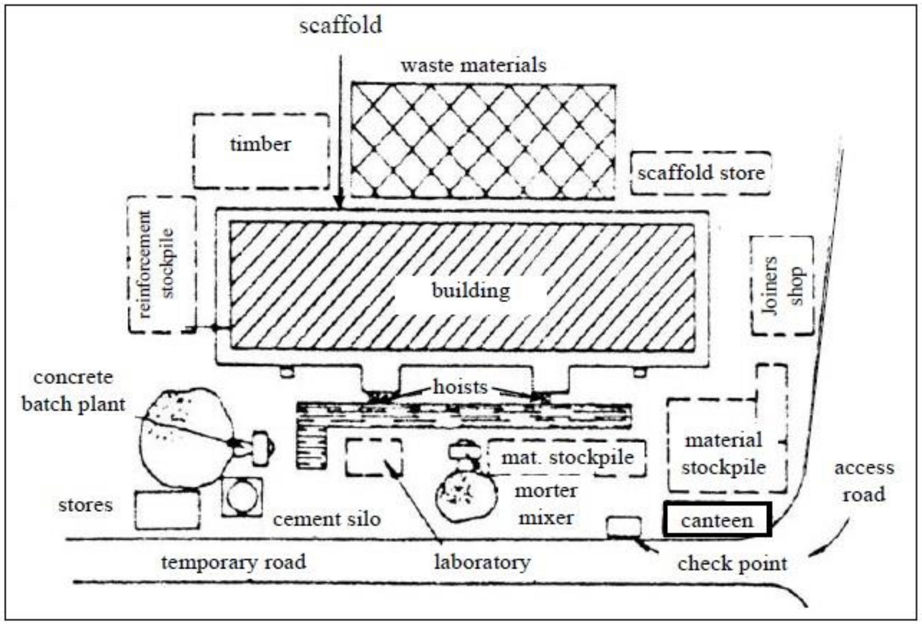
Vehicle routing problem: Illustration



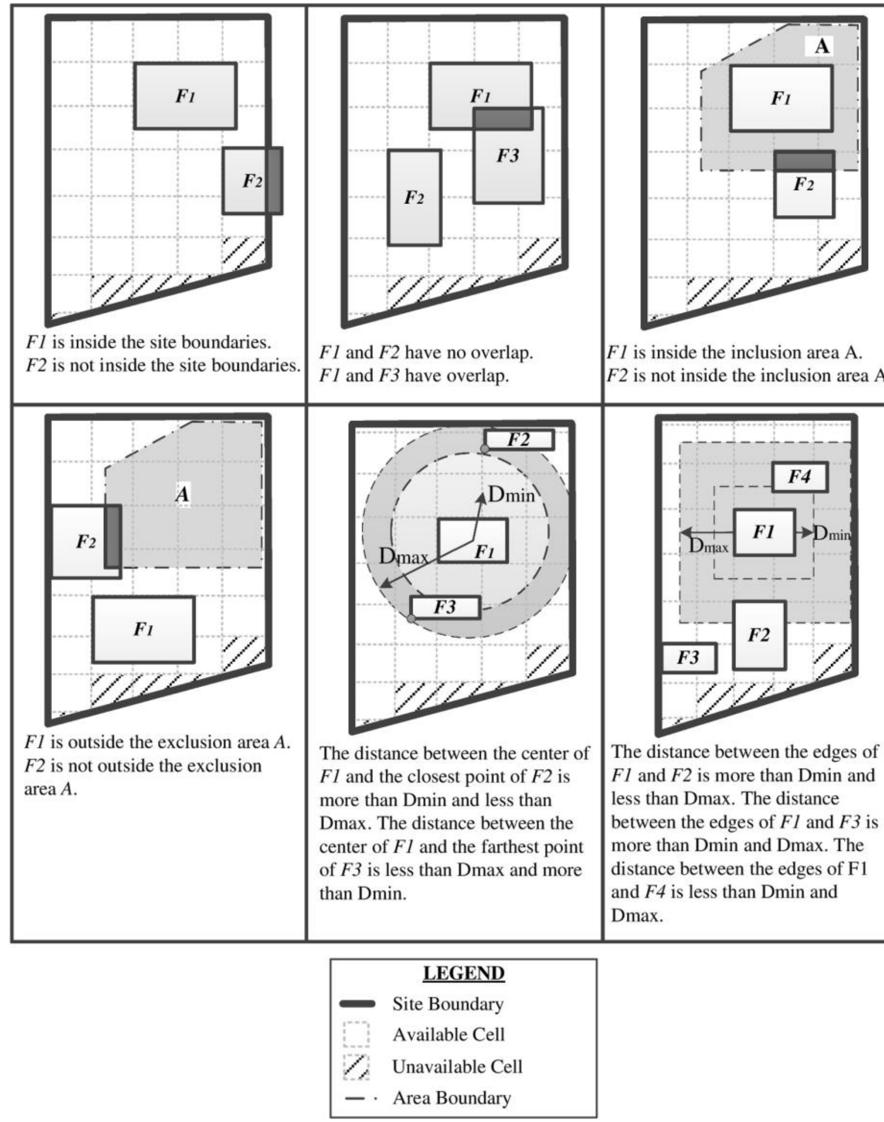
Other problems

- Meta-heuristics are certainly not restricted to combinatorial optimization
- Other problems, with proper formulation, can be solved as well.
- For example: site layout problem
  – the selection of their most efficient layout in order to operate efficiently, cost effectively, and work safely.

Site layout example



https://civilengineeringbible.com



RazaviAlavi and AbouRizk (2017). "Site layout and construction plan optimization using an integrated genetic algorithm simulation framework."

Solution space

- Feasible solution
  - A solution where all the constraints are satisfied
- Feasible region
  - The set of all feasible solutions

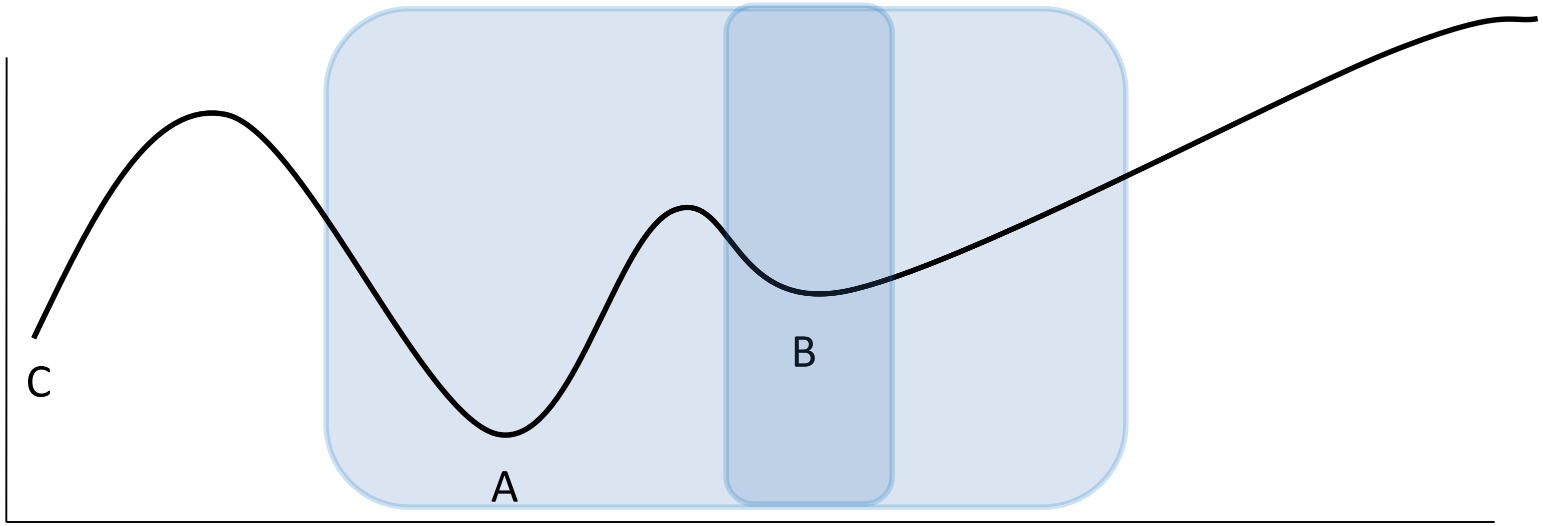
Globally and locally optimal solutions

- A globally optimal solution is one where there are no other feasible solutions with better objective function values.
- A locally optimal solution is one where there are no other feasible solutions "in the vicinity" with better objective function values.
- Note that there may be multiple “globally optimal solutions” which have the same objective values.

Neighborhood of solution

- Neighborhood N(x, ε)
  - *open ball* centered at x: {y: ||y-x|| < ε}, where $\varepsilon > 0$.
  - *closed ball*, with ||y-x|| $\leqq \varepsilon$, where $\varepsilon > 0$.

Locally optimal solutions



- Neighborhood: $N_\varepsilon(f) = \{x\!: x \in F \text{ and } |x - f| \le \varepsilon\}$
- If ε is small enough, B (and C) is locally optimal, if ε gets larger only the global optimum A is locally optimal.

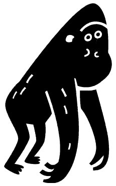
Size of search space

- Taking TSP as an example
- How many possible solutions for a symmetrical TSP ($dis_{ij} = dis_{ji}$)?
- Permutation of *n* numbers except the start is (n-1)!
- A tour can be represented by 2 ways (because of symmetry; 1-2-3-1 is the same as 1-3-2-1)
- The size of search space is (n-1)!/2
- How large is this?

Size of search space (cont.)

- $(n\text{-}1)!/2 \fallingdotseq \frac{1}{2}\sqrt{2\pi(n-1)}\left(\frac{n-1}{e}\right)^{n-1}$
- Suppose we can check all the possible tours for a 20-city problem (6.08E+16 solutions) in 1 hour
- 21-city: 20 hours
- 22-city: 17.5 days
- 25-city: 582 years
- 30-city: 82,973,630 centuries

(human beings appeared on the earth 65,000 centuries ago)



Big O

- Big O: how the size of the input data affects an algorithm's usage of running time or memory.
- It is an asymptotic upper bound for the magnitude of a function in terms of another simpler function.
- For example
  - The number of steps to solve one problem is $T(n) = 5n^2 - 2n + 10$.
  - As n grows large, the n^2 term will dominate, so that all other terms can be neglected
  - We say the complexity is $O(n^2)$

Formal definition

- $O(g(n))= \{ f(n)$: we can find a constant c and n_0 that $0 \leqq f(n) \leqq cg(n)$ for all $n \geqq n_0$
- Using the definition, can we prove that $5n^2 - 2n + 10 = O(n^2)$?
  - Try $c=5$, $n_0 = 5$
  - Obviously, $5n^2 - 2n + 10$ can be expressed as $O(n^3)$, $O(n^4)$, …

Polynomial and exponential complexity

- Polynomial: $O(n^c)$, $c>1$
- Exponential: $O(c^n)$, $c>1$
- Which one grows faster?

n	n^2	2^n	n!
2	4	4	2
10	100	1024	3628800
50	2500	1.13E+15	3.04E+64
150	**22500**	**1.43E+45**	**5.7E+262**

$O(\log n) < O(n^{1/2}) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$

Example: Bubble sorting

- Bubble sorting: repeatedly comparing two items at a time and swapping them if they are in the wrong order.
- How many comparison need to be made to sort a list of number?
- Need $(n\text{-}1)+(n\text{-}2)+\ldots 1 = n(n\text{-}1)/2$ comparisons
- Complexity is $O(n^2)$

Example: Binary search

- Guess a positive integer, between 1 and n, selected by another player, using only questions answered with yes or no:
  is the target smaller than a certain value?
- How many questions are required, at most, to find the answer?
- Worst case: $\log_2(n)$

P and NP

- Problems can be divided into two categories: those for which there exists an algorithm to solve it with polynomial time complexity, and those for which there is no such algorithm *but there is also no proof that no such algorithm exists*.
- We denote the former class of problems by P.
- NP: the class of problems which can be solved by a **non-deterministic polynomial** algorithm.

Non-deterministic polynomial algorithm

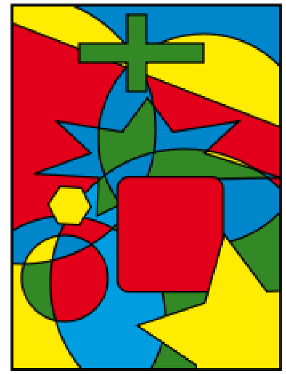
- Non-deterministic polynomial algorithm includes (1) guessing and (2) checking
- A problem may be hard to solve but easy to check the correctness once we are given a solution
  - Find a solution for $X^5-2X^2-3X+1=0$
  - Is X=1 is a solution? how about 2?
  - Given above, which will the be the next value to “guess”?
- Another example: encryption
  - It may be hard to find the key for encryption, but once you find it, you can easily check whether it works

NP-hard and NP-complete

- P is a subset of NP
- **NP-complete (NPC)**: an NP problem for which it is possible to reduce other NP problems to it in polynomial time.
- **NP-hard**: problem X is NP-hard if there is an NP-complete problem Y such that Y is reducible to X in polynomial time.
- Reducing problem A to another problem B means describing an algorithm to solve problem A under the assumption that an algorithm for problem B already exists (using it as a subroutine).

Examples of NP problem

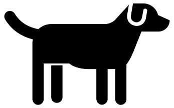
- Knapsack problem
- Travelling salesman problem
- Graph coloring problem
  - Optimal assignment of "colors" to certain vertices in a graph such that no two adjacent vertices share the same color



NP Issues

- Up to now, none of the NPC problems can be solved by a deterministic polynomial time algorithm in the worst case.
- It does not seem to have any polynomial time algorithm to solve the NPC problems.
- The lower bound of any NPC problem seems to be in the order of an exponential function.
- An NP problem is not necessarily "difficult".

Analogy

- "If I own a dog, it can speak fluent English." 
- We cannot prove that dogs do not speak English, even though no one has ever heard a dog speaks English
- But no one in their right mind should believe dogs speak English, so they would conclude that I do not own a dog
- Similarly, if a problem is, no one in their right mind should believe it can be solved in polynomial time.

Team members

- Submit the membership of your team to the Moodle System before 11:00pm, March 12
(maximum number of teammates is 5)
  - Name of your team
  - Student IDs
  - Students' Names