Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

MSE Algorithms

L02: Constructive Methods

Samuel Beer

Course Overview

1. Introduction: Basic problems and algorithms
2. **Constructive methods: Random building, Greedy**
3. Local searches
4. Randomized methods
5. Threshold accepting, Simulated annealing
6. Decomposition methods: Large neighborhood search
7. Learning methods for solution building: Artificial ant systems
8. Learning methods for solution improvement: Tabu search
9. Methods with a population of solutions: Genetic algorithms

Lecture 2:
Constructive Methods

Goal:

Know different simple methods to find a first «reasonably good» solution.

Topics:

- Random Sampling
- Greedy Constructions
- Exhaustive Search
- Lookahead: Pilot Methods and Beam Search
- Santa Challenge

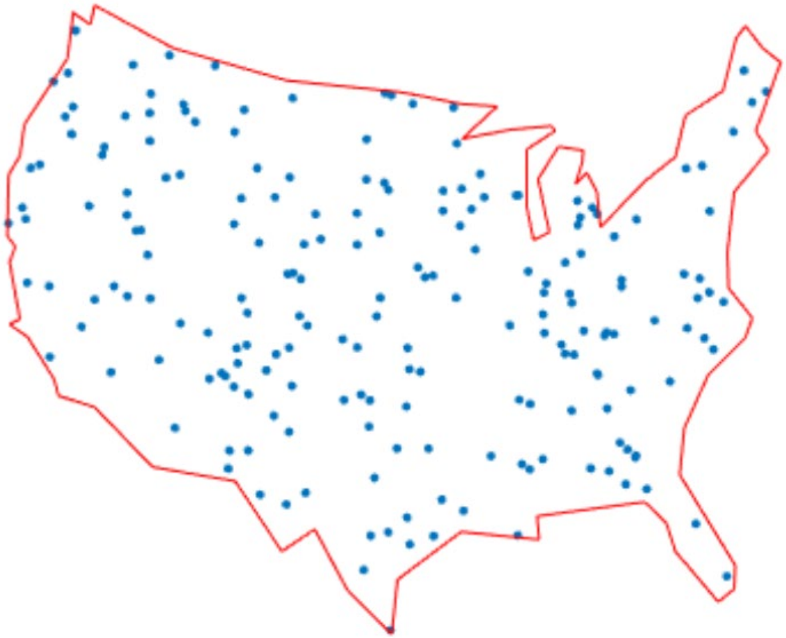
Recap: Traveling Salesperson Problem TSP

Input: n cities, $D = (d_{ij})$ distances matrix between cities i and j.

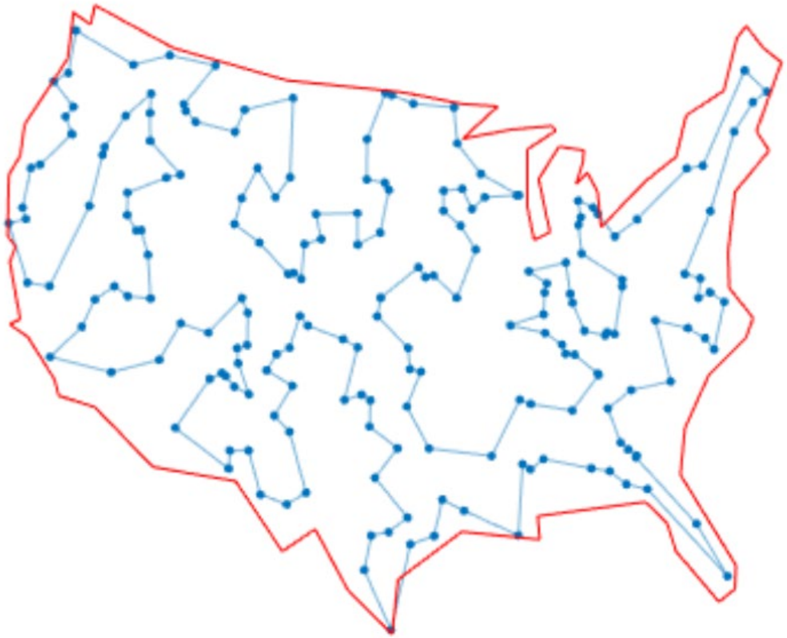
Problem: Find the shortest tour passing exactly once in each city.

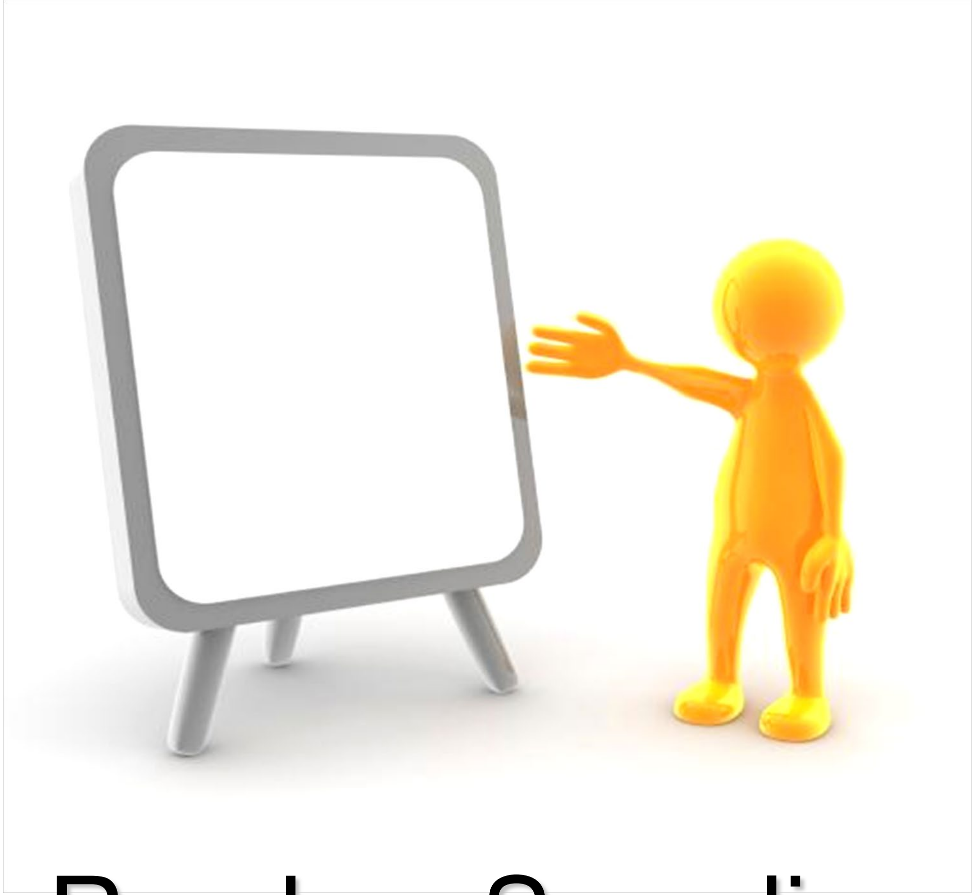
SOLUTION is a sequence of cities p_1, … p_n with minimum cost $\sum_{i=1}^{n-1} d_{p_i p_{i+1}} + d_{p_n p_1}$

Problem Input



One "Good" Solution





Random Sampling

Meta-Heuristic: Random Sampling

[also known as "Random Building"]

Idea: Generate a solution randomly, uniformly in the solution space

Advantages

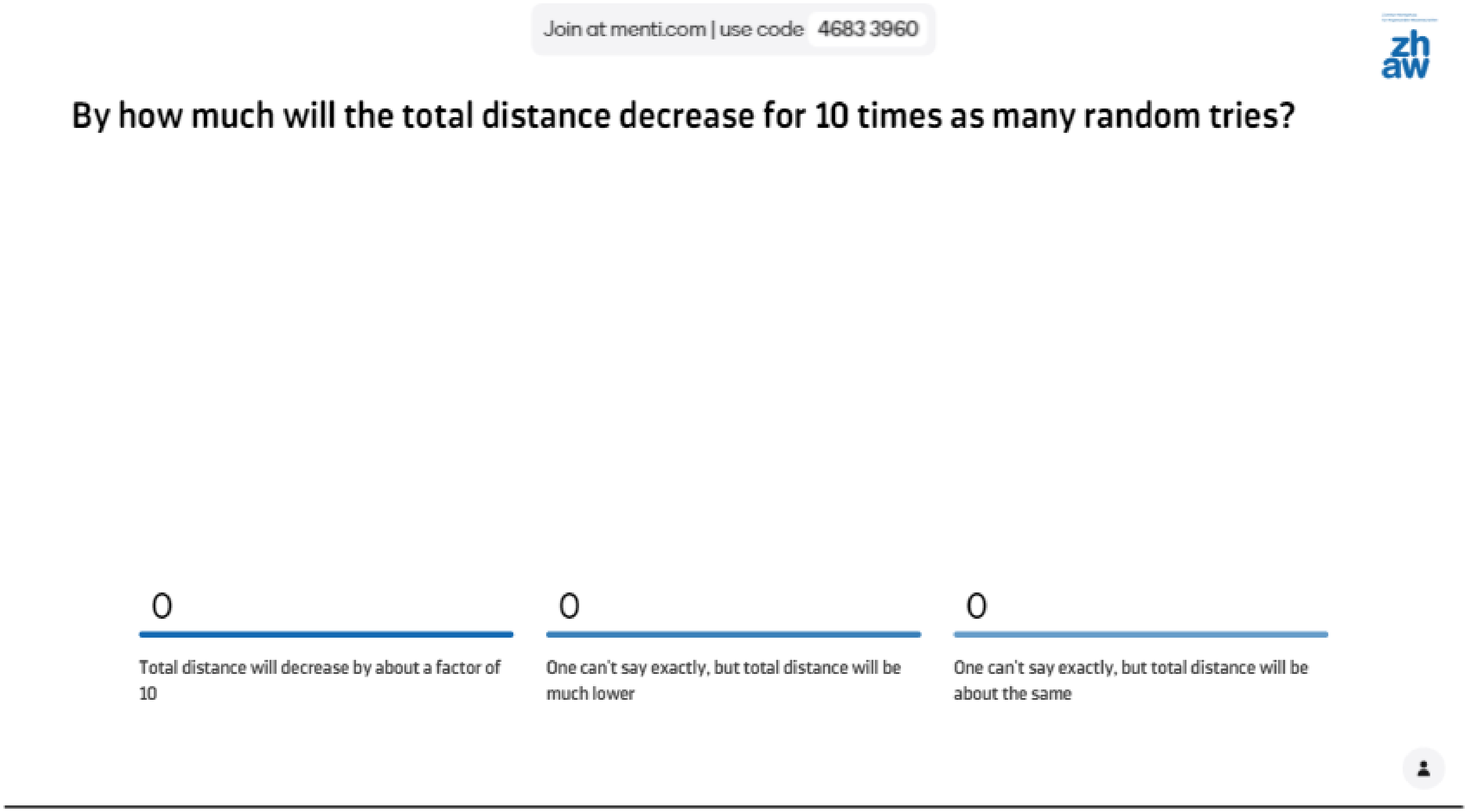
- One of the simplest methods
- Applicable for almost all problems
- Easy to implement

Disadvantages

- The solution’s quality can be very bad
- (A uniform generation of solutions sometimes is not trivial if there are constraints which have to be respected in addition to the cost function)

Example TSP: Generate a random permutation of *n* elements

Random Sampling for TSP



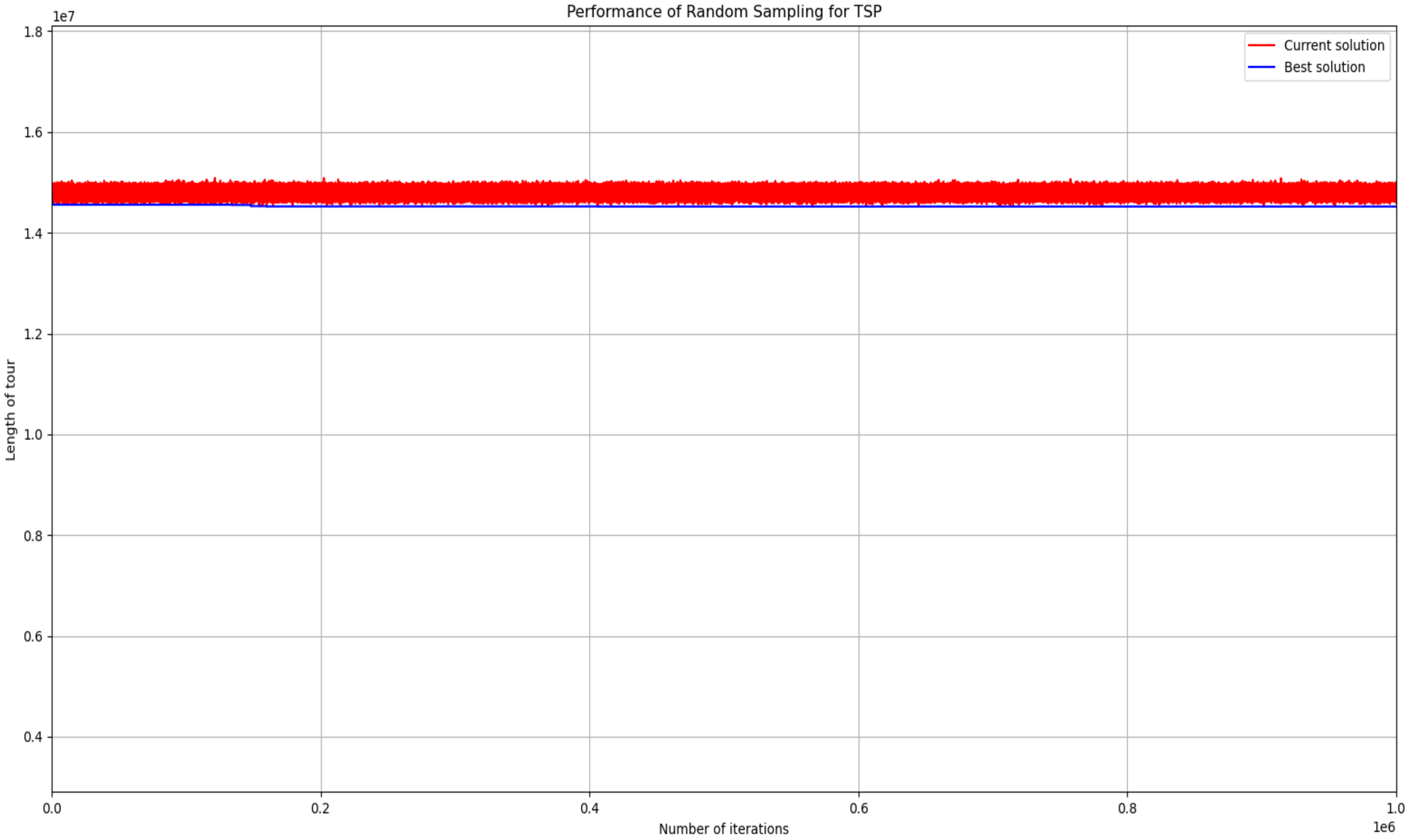
Join at menti.com | use code 4683 3960

By how much will the total distance decrease for 10 times as many random tries?

0	0	0
Total distance will decrease by about a factor of 10	One can't say exactly, but total distance will be much lower	One can't say exactly, but total distance will be about the same

Random Sampling for TSP (Conclusion)

Random Sampling



Greedy Construction

Meta-Heuristic: Greedy Construction

Idea: Build a solution, element by element, by systematically adding the most appropriate ("best") element with reference to some criterion. A choice taken at some step is never questioned later on.

Details:

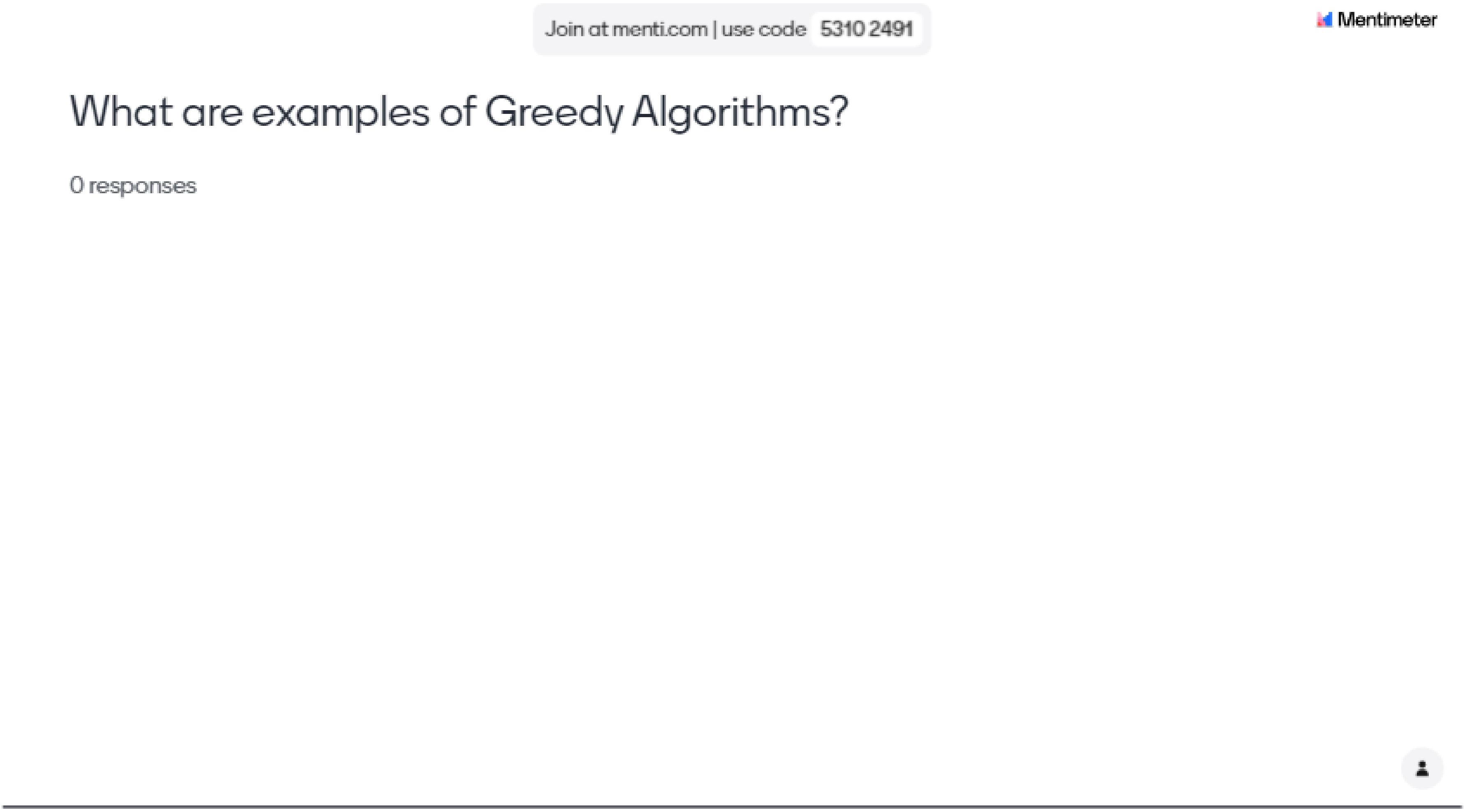
- The method starts with a partial solution *S* which is empty or trivial to construct
- Maintains a list *R* of elements that might be added to *S* thus expanding it
- Needs a cost function *c*(*S, e*) that measures the quality of adding element *e* to the partial solution *S*
- Adding *e* to *S* generally implies restrictions for the remaining elements in *R*

Examples:

- TSP: Start with a first city, always add one new "good" edge
- Vertex Colouring: Start with a vertex, always select a new vertex and give it a valid color

Remarks: This works optimally for some problems, and sub-optimal for other problems

Question



Join at menti.com | use code 5310 2491

Mentimeter

What are examples of Greedy Algorithms?

0 responses

Greedy Construction: Basic Algorithm

```
Build a minimal partial solution $S$        // often an empty solution
Initialize $R$                              // initial set of elements that may be added to $S$

Repeat
    Evaluate $c(S, e)$ for each $e \in R$
    Choose $e'$ which optimizes $c(S, e)$
    Add $e'$ to the partial solution $S$
    Remove from $R$ all elements that may not be added to $S$ any more
Until $S$ is a complete solution
```

Nearest Neighbor for TSP

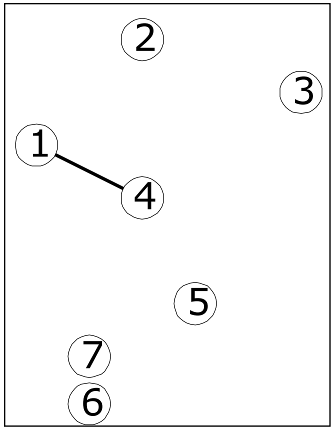
Nearest Neighbor of Last City

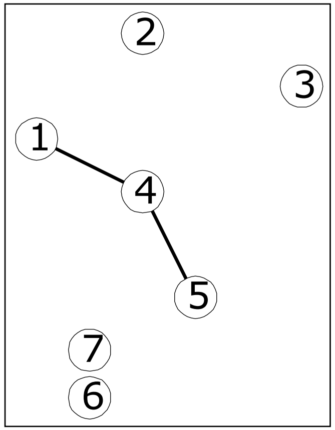
- *S* = [1]   a tour starting in city 1; S will be extended throughout the algorithm

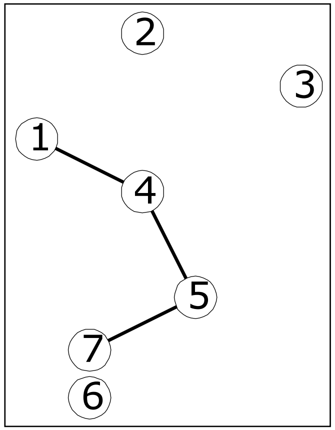
- *R*: set of cities not yet visited; initially all cities except city 1; empty in the end

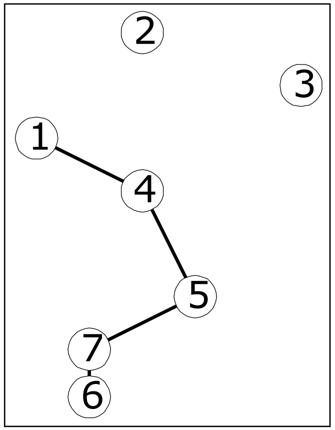
- *c*(*S*, *e*) = distance from the last city in *S* to city *e*

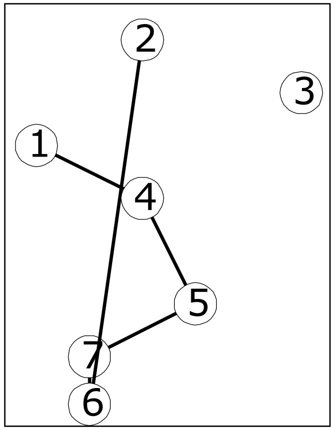
TSP: Application of Nearest Neighbor

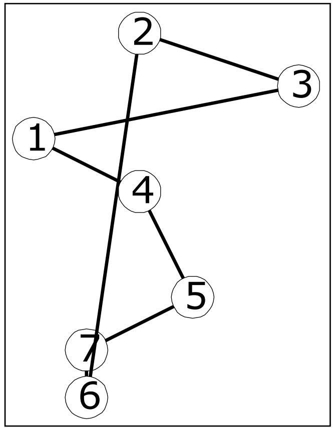








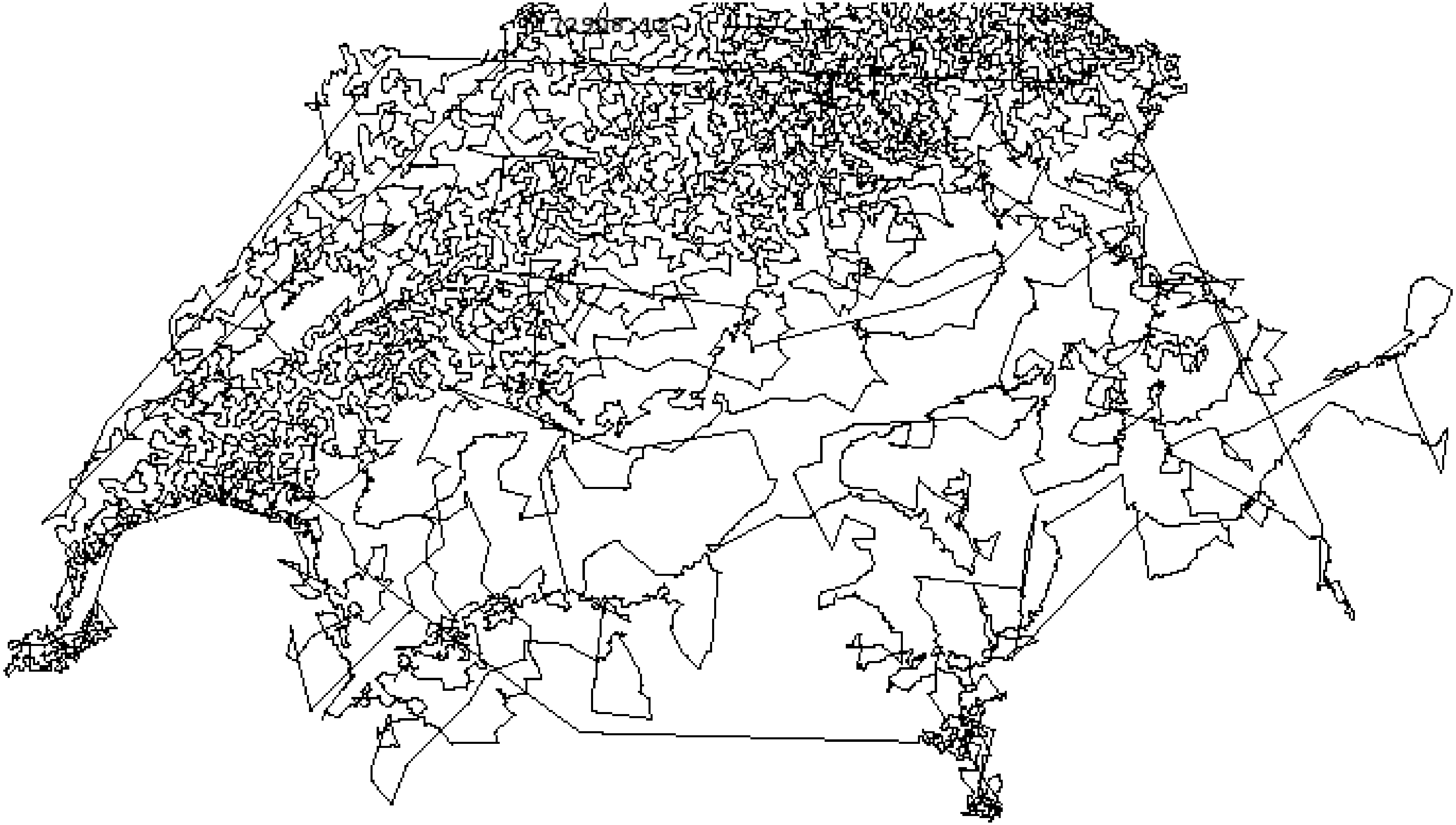




Issues:

- Cities may be skipped in the beginning
- The tour degrades during the last iterations

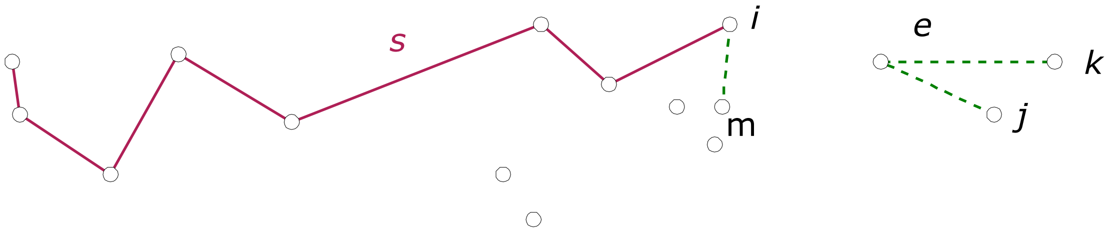
TSP: Nearest Neighbor for Swiss Road Crossings



Other Greedy Strategies for TSP (1)

Best Global Edge

- Initial S : $\emptyset$
- R : Set of edges that can be added to S such that:
  - No cycle is created
  - No vertex with degree > 2 is created
- $c(S, e)$ = weight of edge e (this is independent from S)



Maximum Regret

- Initial $S = \{1\}$
- R : Set of cities not yet visited (+ city 1, if all cities were already visited)
- $c(S, e)$ = Regret of **not** going to e from i, the last city of tour S:

$$c(S, e) := \min_{j,k \in R} (d_{je} + d_{ek}) - \min_{r \in R} (d_{ie} + d_{er})$$

- Choose the largest $c(S, e)$

Other Greedy Strategies for TSP (2)

Random Best Insertion

- *S* = Tour on 2 cities (e.g. 1—2—1, two random first cities)
- *R*: Set of cities not yet visited; Choose $e \in R$ randomly in every step
- *c*(*S*, *e*) = Minimum insertion cost of city *e* between 2 cities of partial tour *S*
- Choose the smallest *c*(*S*, *e*)

Most Distant Best Insertion

- Same as Random Best Insertion, but always choose *e* furthest from the partial tour *S* in every step

Nearest Best Insertion

- Same as Random Best Insertion, but always choose *e* nearest from the partial tour *S* in every step

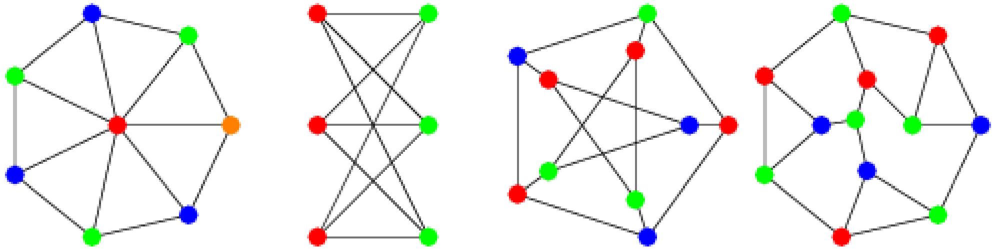
Greedy Construction for Vertex Coloring

Vertex Coloring

- A **vertex coloring** is an assignment of labels or colors to each vertex of a graph such that no edge connects two identically colored vertices.

- The most common type of vertex coloring seeks to *minimize* the number of colors for a given graph.

- Such a coloring is known as a minimum vertex coloring, and the minimum number of colors which with the vertices of a graph may be colored is called the chromatic number, denoted $X(G)$.



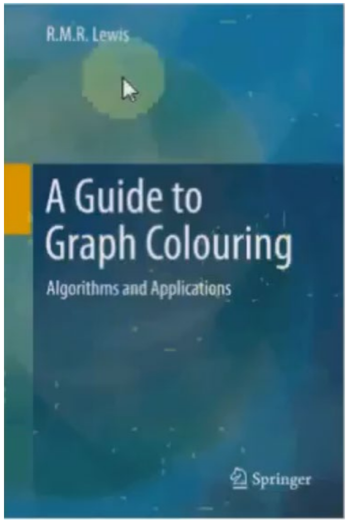
http://mathworld.wolfram.com/VertexColoring.html

Greedy Strategies for Vertex Coloring

First Fitting Color

- Select an ordering of the vertices
- $s = \emptyset$ // Set of vertices already coloured
- $R = V$ // Set of vertices not coloured yet
- $C(s, e)$ = Set of colors that can be assigned to vertex e without violation of the coloring rule, sorted **lexicographically**
- Choose the first color in $C(s, e)$

Note: For every graph, there exists an ordering of its vertices such that the greedy algorithm "First Fitting Color" produces an optimal solution.



R.M.R. Lewis

A Guide to Graph Colouring

Algorithms and Applications

Springer

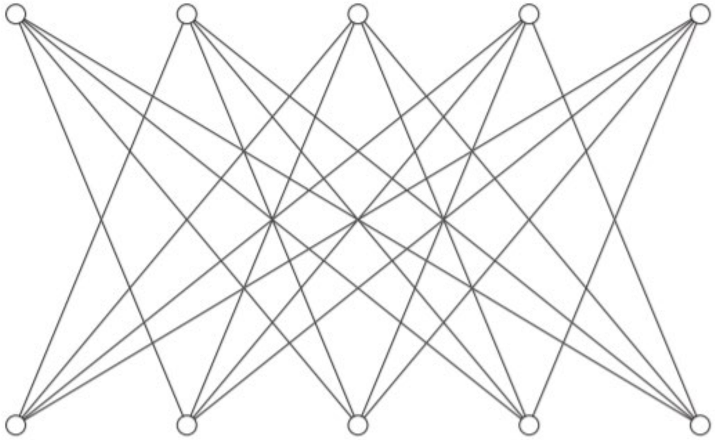
Exercise

Vertex Coloring on Bipartite Graphs

Given an almost complete bipartite graph G with *2k* vertices u_1, .. u_k and v_1, … v_k, and all edges (u_i, v_j) with $1 \leq i \neq j \leq k$.

We apply the greedy strategy "First Fitting Color" for Vertex Coloring.

How many colors does the algorithm need?



SOLUTION: Vertex Coloring

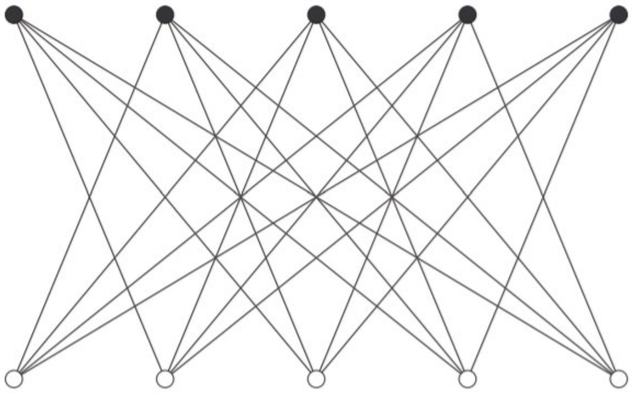
Vertex Coloring on Bipartite Graphs

The number of colors depends on the ordering of the vertices:

1. Case: Vertices are ordered

$v_1, v_2, \ldots, v_k, u_1, u_2, \ldots, u_k$

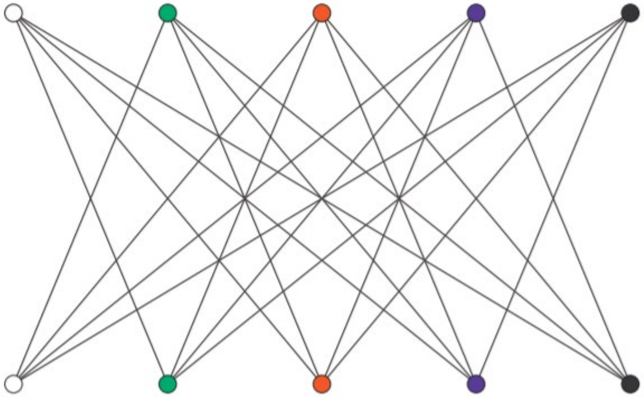
Then the algorithm colors the graph with 2 colors.



2. Case: Vertices are ordered

$v_1, u_1, v_2, u_2 \ldots, u_k$

Then the algorithm colors the graph with *k* colors.



More Greedy Strategies for Vertex Coloring

Decreasing Degree

- Sort the vertices by decreasing order of degree
- Remaining is identical to First Fitting Color

D_{SATUR} (Degree of Saturation)

- $s = \varnothing$ // Vertices already coloured
- R = V // Set of vertices not coloured yet
- Choose the uncolored vertex with the highest "saturation degree", i.e. with the maximum number of different adjacent colors. Break ties by choosing the vertex with maximum degree.
- Selection rule can be expressed as a formula:
  $c(s, e)$ = degree of e + |V| · number of different colours used by already colored vertices that are connected to e

More Constructive Methods

Exhaustive Search

Idea: Generate all feasible solutions and find the optimum one

Advantages

- Guarantees to find an optimal solution
- Often easy to implement

Disadvantages

- For many problems, the set of feasible solutions is exponential, thus, running time is too high.

Example TSP:

- Generate all permutations of the *n* cities, and calculate their costs.

Exhaustive Search as Constructive Strategy

Idea

- Maintain a set of partial solutions *P*
- Starts with P = {s}, where *s* is the empty solution
- In each step
  - for each $p \in P$, compute **all** elements *e* that might be added to the partial solution *p* to form a new partial solution;
  - remove *p* from *P* and add $p \cup \{e\}$ to *P*
- Compute the costs for all solutions in *P*

Meta-Heuristic: Pilot Method

Weakness of greedy building methods:

- Too short-sighted
- A seemingly good choice at a given step can globally be disastrous

Idea of Pilot Method

- Evaluate the quality of adding an element *e* to *s* by completing it *to a full solution*

```
For all elements $e$ that can be added to $s$
    Complete $s + e$ into a full solution by
        some heuristic (the "pilot")
    Keep the element leading to the best
        complete solution
```

Remarks

- The time complexity is increased by applying this method to each partial solution
- The pilot-heuristic to complete a solution must be chosen, e.g. this could be a greedy method, greedy method + local search, etc.

Exercise

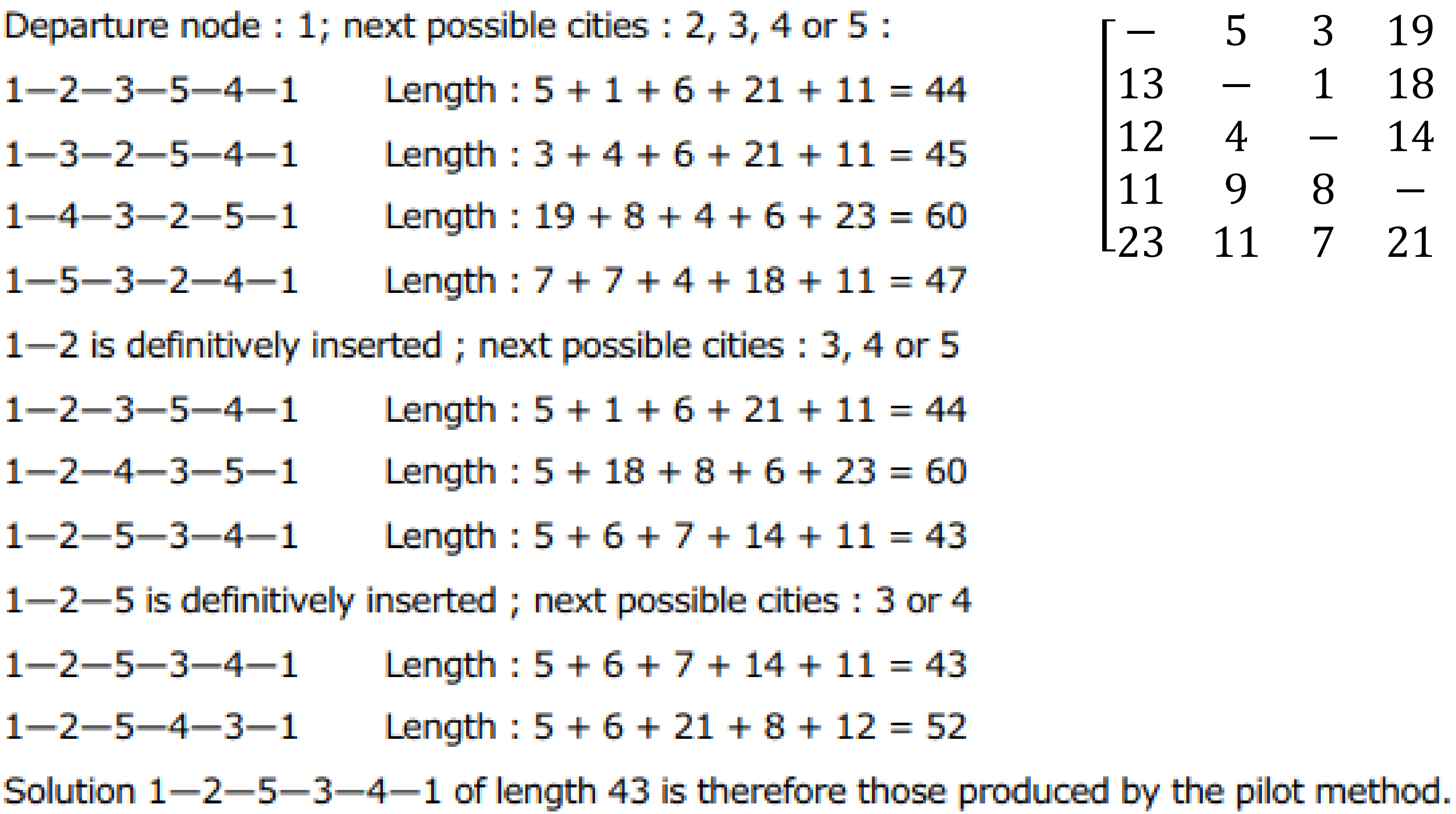
Pilot Method

Apply the Pilot Method to the TSP instance below, given by its distances matrix. Use the Nearest Neighbour heuristic as pilot strategy. The tour starts with the first city.

$$\begin{bmatrix} - & 5 & 3 & 19 & 7 \\ 13 & - & 1 & 18 & 6 \\ 12 & 4 & - & 14 & 6 \\ 11 & 9 & 8 & - & 10 \\ 23 & 11 & 7 & 21 & - \end{bmatrix}$$

SOLUTION: Pilot Method

Pilot Method



Departure node : 1; next possible cities : 2, 3, 4 or 5 :

1—2—3—5—4—1 Length : 5 + 1 + 6 + 21 + 11 = 44

1—3—2—5—4—1 Length : 3 + 4 + 6 + 21 + 11 = 45

1—4—3—2—5—1 Length : 19 + 8 + 4 + 6 + 23 = 60

1—5—3—2—4—1 Length : 7 + 7 + 4 + 18 + 11 = 47

1—2 is definitively inserted ; next possible cities : 3, 4 or 5

1—2—3—5—4—1 Length : 5 + 1 + 6 + 21 + 11 = 44

1—2—4—3—5—1 Length : 5 + 18 + 8 + 6 + 23 = 60

1—2—5—3—4—1 Length : 5 + 6 + 7 + 14 + 11 = 43

1—2—5 is definitively inserted ; next possible cities : 3 or 4

1—2—5—3—4—1 Length : 5 + 6 + 7 + 14 + 11 = 43

1—2—5—4—3—1 Length : 5 + 6 + 21 + 8 + 12 = 52

Solution 1—2—5—3—4—1 of length 43 is therefore those produced by the pilot method.

$$\begin{bmatrix} - & 5 & 3 & 19 & 7 \\ 13 & - & 1 & 18 & 6 \\ 12 & 4 & - & 14 & 6 \\ 11 & 9 & 8 & - & 10 \\ 23 & 11 & 7 & 21 & - \end{bmatrix}$$

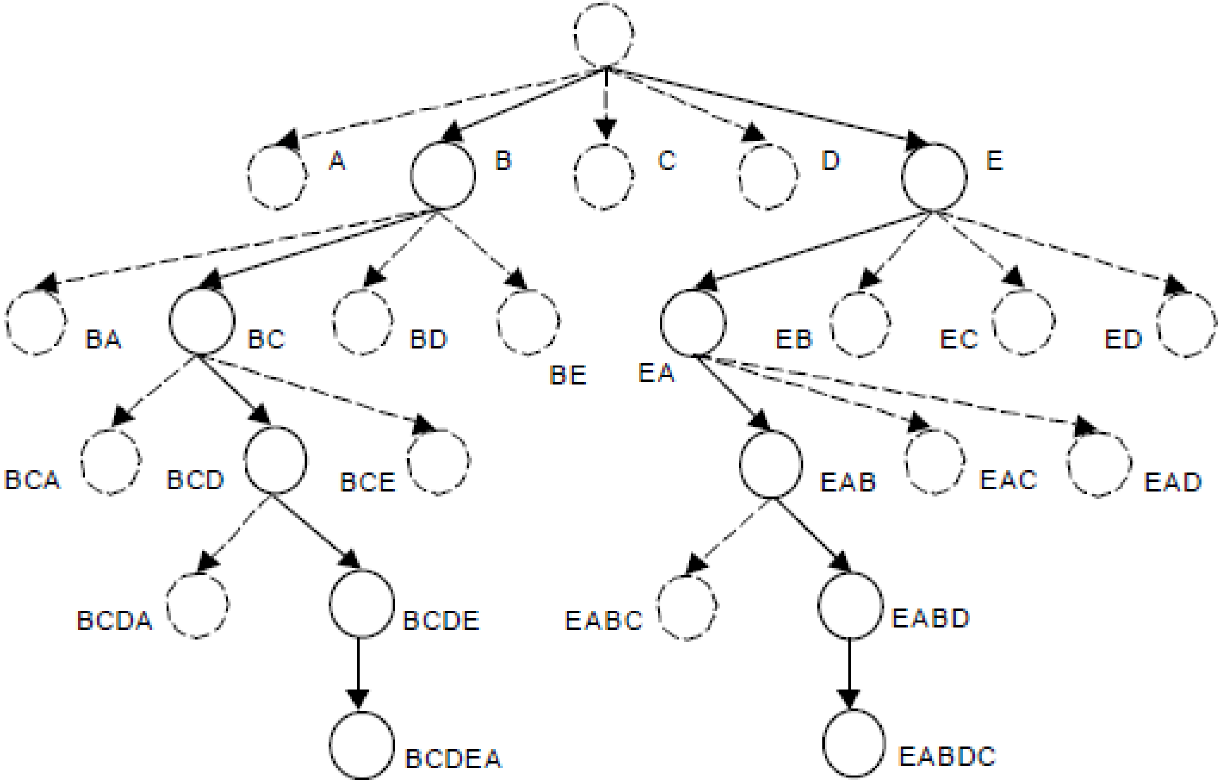
Meta-Heuristic: Beam Search

Idea

- Similar to Exhaustive Search, but maintains only a "small" set of partial solutions *P*
- More precisely, define a constant *B* ("beam width") and ensure |P| ≤ *B* in each step of the algorithm
- In each step:
  - Keep the B most promising partial solutions and store them in P
  - *Extension*: for each partial solution p in P, expand the solution up to depth *k* to explore how good each partial solution might be (k=1 is no lookahead).
- Note: With an infinite beam width, no states are pruned and beam search is identical to breadth-first search

Beam Search Example

Beam Search with beam width B=2 and browse depth k = 1.



Exercise

Beam Search

Apply the Beam Search heuristic to the TSP instance below, given by its distances matrix. Use the parameters *B*=2 and *k*=2. Only add one city to the retained B partial tours per step. The tour starts with the first city.

$$\begin{bmatrix} - & 5 & 3 & 19 & 7 \\ 13 & - & 1 & 18 & 6 \\ 12 & 4 & - & 14 & 6 \\ 11 & 9 & 8 & - & 10 \\ 23 & 11 & 7 & 21 & - \end{bmatrix}$$

SOLUTION: Beam Search

Beam Search

$$\begin{bmatrix} - & 5 & 3 & 19 & 7 \\ 13 & - & 1 & 18 & 6 \\ 12 & 4 & - & 14 & 6 \\ 11 & 9 & 8 & - & 10 \\ 23 & 11 & 7 & 21 & - \end{bmatrix}$$

First step: 1 - … ?

1 - 2 - 3	5 + 1 = 6
1 - 2 - 4	5 + 18 = 23
1 - 2 - 5	5 + 6 = 11
1 - 3 - 2	3 + 4 = 7
1 - 3 - 4	3 + 14 = 17
1 - 3 - 5	3 + 6 = 9
1 - 4 - 2	19 + 9 = 28
1 - 4 - 3	19 + 8 = 27
1 - 4 - 5	19 + 10 = 29
1 - 5 - 2	7 + 11 = 18
1 - 5 - 3	7 + 7 = 14
1 - 5 - 4	7 + 21 = 28

Second step: 1 - 2 - … ?
1 - 3 - … ?

1 - 2 - 3 - 4 1 - 2 - 3 - 5 1 - 2 - 4 - 3 1 - 2 - 4 - 5 1 - 2 - 5 - 3 1 - 2 - 5 - 4	5 + 1 + 14 = 20 5 + 1 + 6 = 12 5 + 18 + 8 = 31 5 + 18 + 10 = 33 5 + 6 +7 = 18 5 + 6 + 21 = 32
1 - 3 - 2 - 4 1 - 3 - 2 - 5 1 - 3 - 4 - 2 1 - 3 - 4 - 5 1 - 3 - 5 - 2 1 - 3 - 5 - 4	3 + 4 + 18 = 25 3 + 4 + 6 = 13 3 + 14 + 9 = 26 3 + 14 + 10 = 27 3 + 6 + 11 = 20 3 + 6 + 21 = 30

Third step: 1 - 2 - 3 - … ?
1 - 3 - 2 - … ?

1 - 2 - 3 - 4 - 5 1 - 2 - 3 - 5 - 4	5 + 1 + 14 + 10 = 30 5 + 1 + 6 + 21 = 33
1 - 3 - 2 - 4 - 5 1 - 3 - 2 - 5 - 4	3 + 4 + 18 + 10 = 35 3 + 4 + 6 + 21 = 34

Last step: 1 - 2 - 3 - 4 - … ?
1 - 2 - 3 - 5 - … ?

1 - 2 - 3 - 4 - 5 - 1	5 + 1 + 14 + 10 + 23 = 53
1 - 2 - 3 - 5 - 4 - 1	5 + 1 + 6 + 21 + 11 = 44

Lecture 2: Summary

Meta-Heuristics for Optimization Problems:

1. **Random Sampling** are easy to implement and can generate solutions very fast, but usually with bad quality

2. **Greedy Methods** can create solutions with reasonable good quality (e.g. Random Best Insertion for TSP). But greedy methods can also fail desastrously

3. **Look-Ahead Methods** (e.g. Pilot Method, Beam Search) can improve greedy methods

Exercises and Santa Challenge

♫ Alarm bells ring, are you listening? Santa's sleigh has gone missing ♫