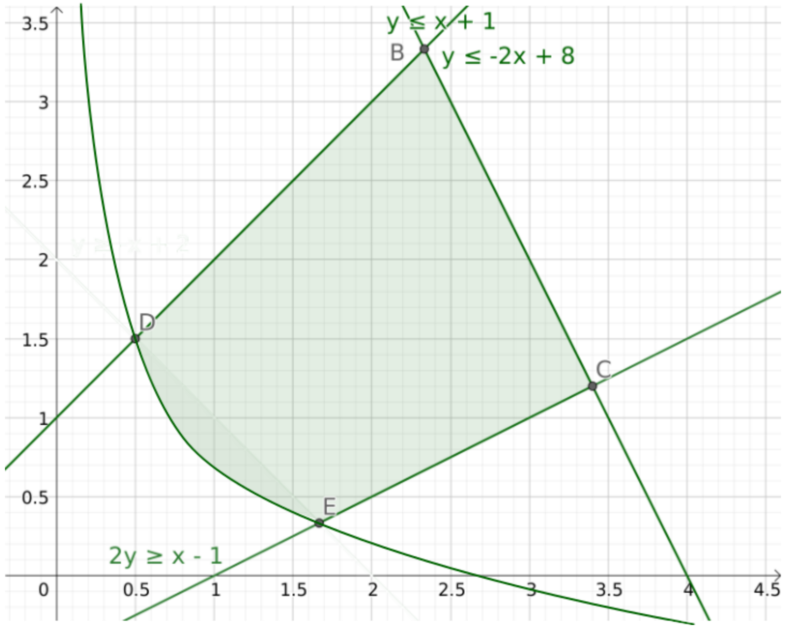
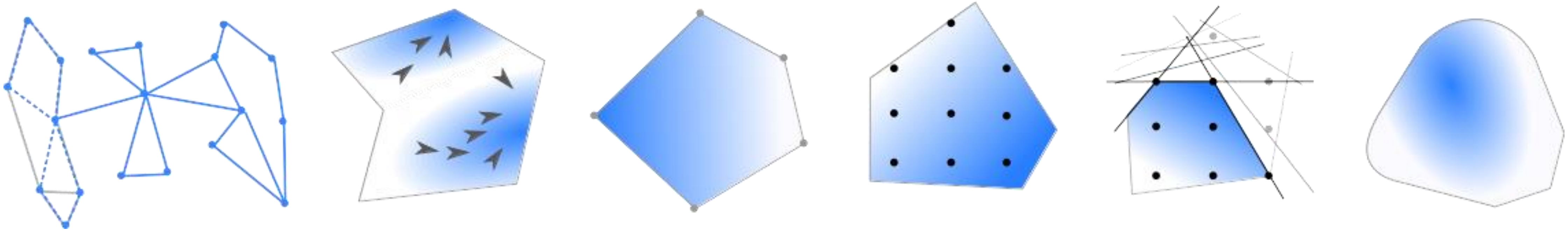
Meta-Heuristics 2

COMP4691 / 8691





Previously on COMP4691(8691)

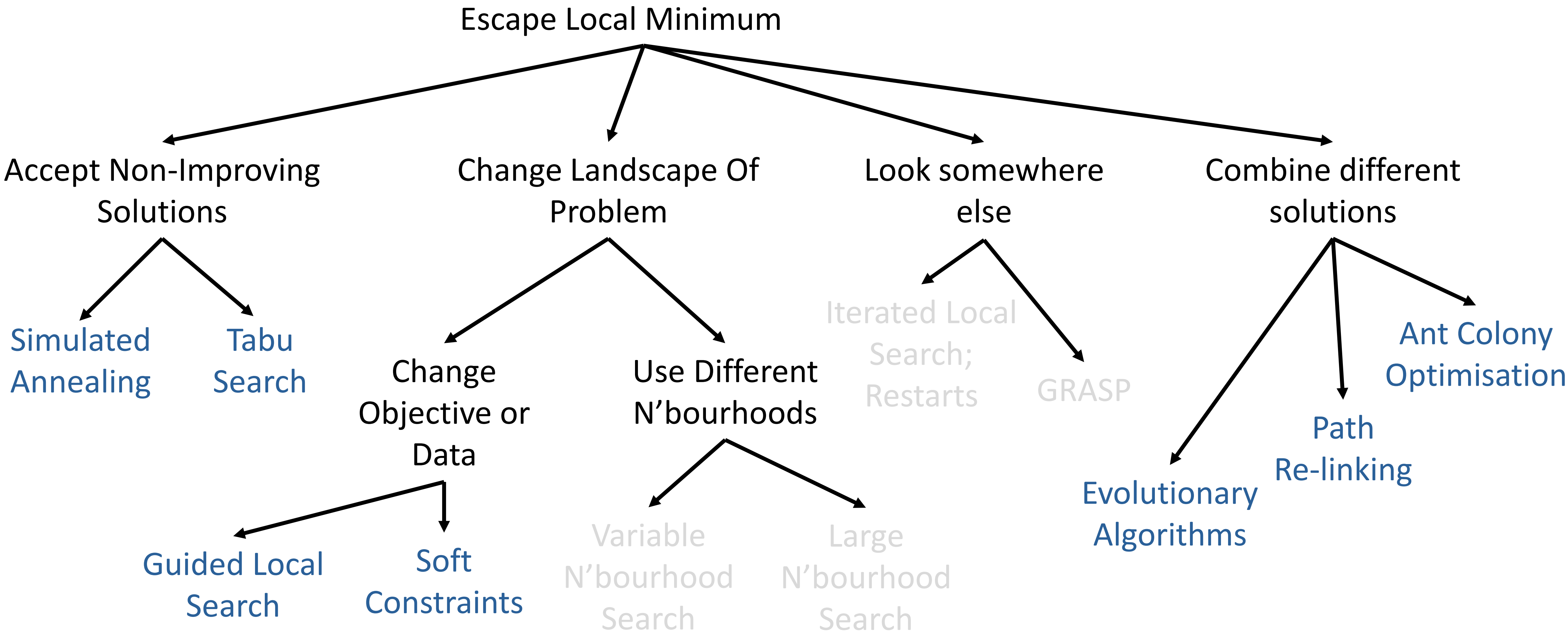
- (Stochastic) Local Search
  - … and how Local Search gets stuck in local minima

- Some Metaheuristic to escape local minima
- Based on
  - Accepting non-improving solutions
  - Changing the objective or the data
  - Combining Solutions

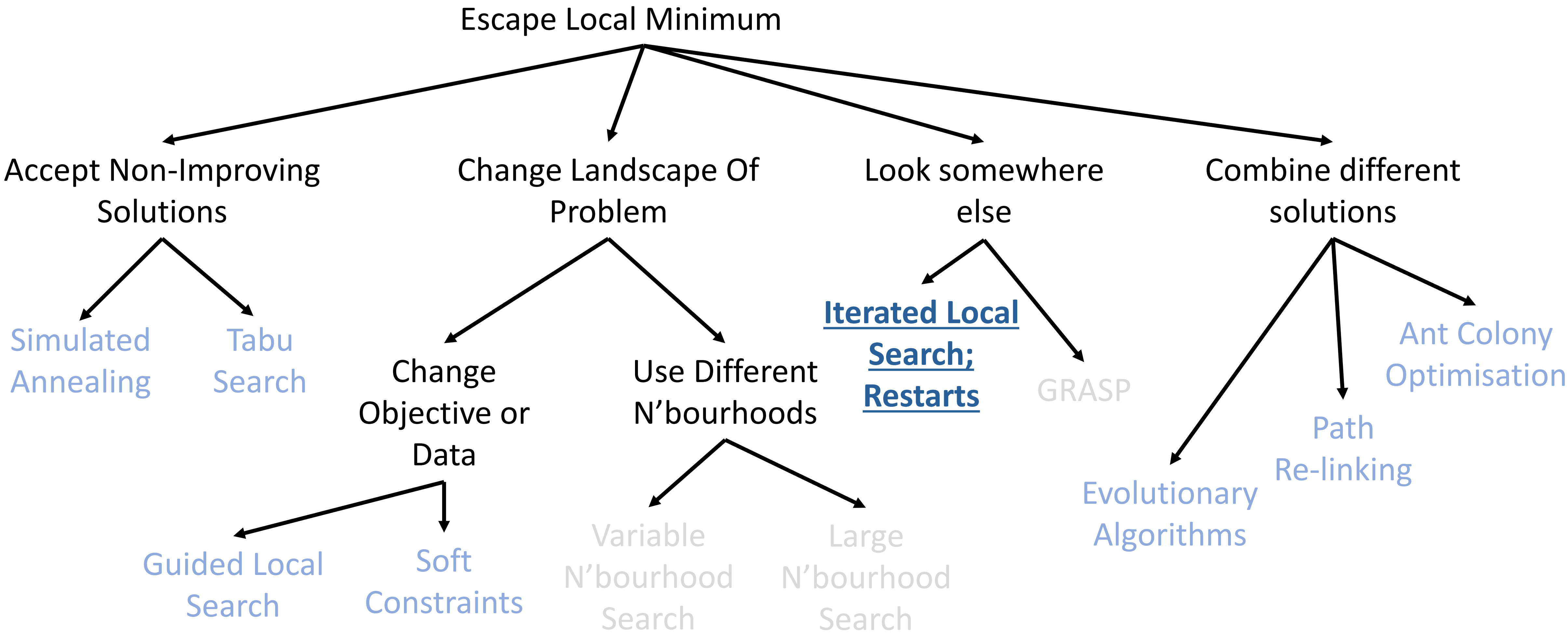
Today:

- More Metaheuristics!

Meta-heuristics



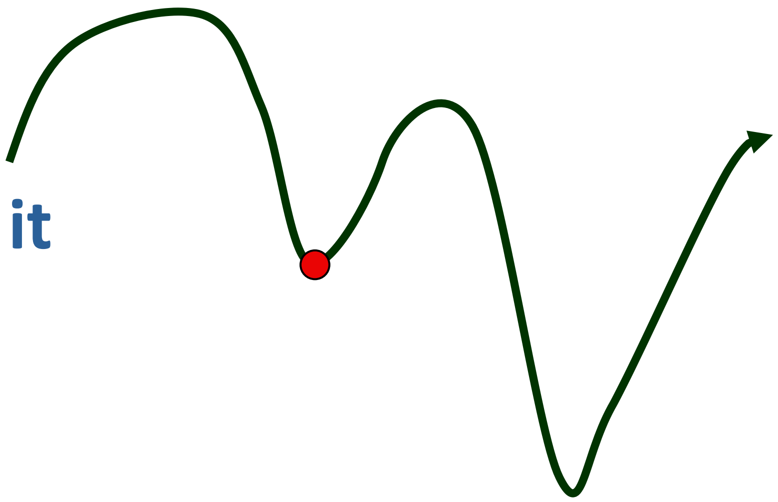
Meta-heuristics: An Incomplete Survey



Iterated Local Search

- Closely related to Tabu Search
- **Instead of climbing up the wall, it is kicked over it**

- Find Local minimum
- Repeat
  - Perturb the solution (problem dependent)
    - = fix part of the solution, randomise the rest
  - Find Local minimum
  - Accept Solution?



Iterated Local Search

- Perturbation
  - A.K.A “Kick” or “Shake”
  - “Strength” = how many elements to change
- Similar to Tabu Search
  - Fix too much → Fall back to same solution
  - Fix too little → Same as random restart
- Can store some solution history
  - Use it to control perturbation

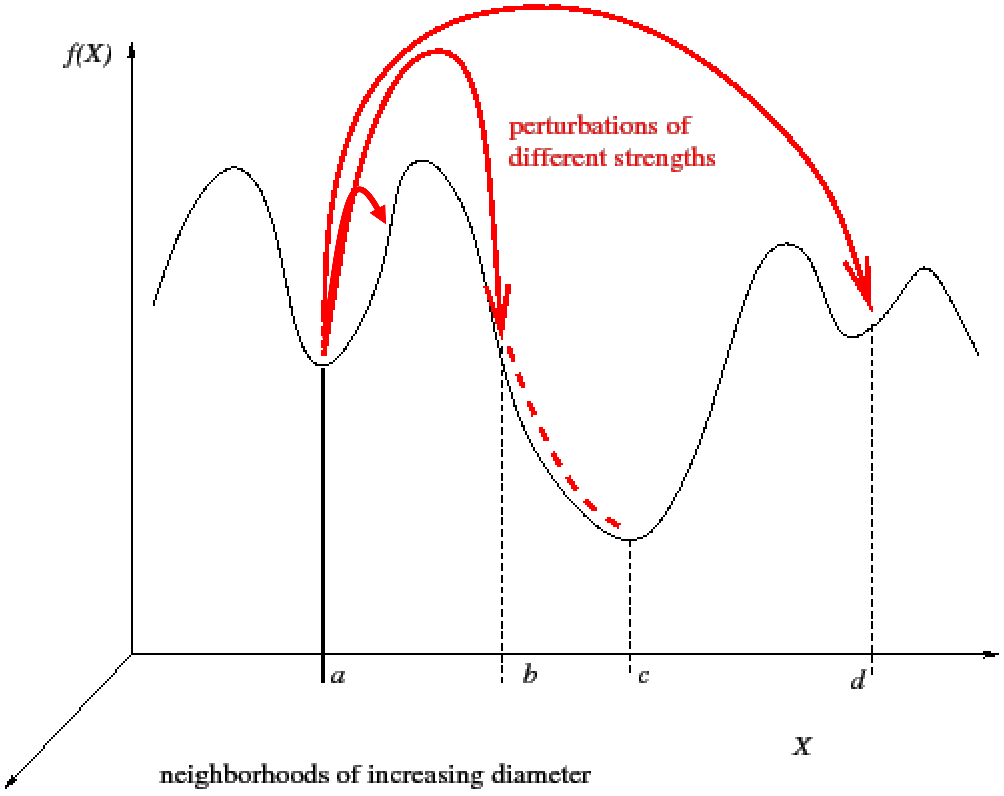


Diagram credit: Wikipedia

Iterated Local Search

TSP

- Good perturbation is “Double Bridge”
- = replace 4 arcs

- Good because
  - it is hard for local search to undo
  - a good initial tour will still be good (only small increase in objective)

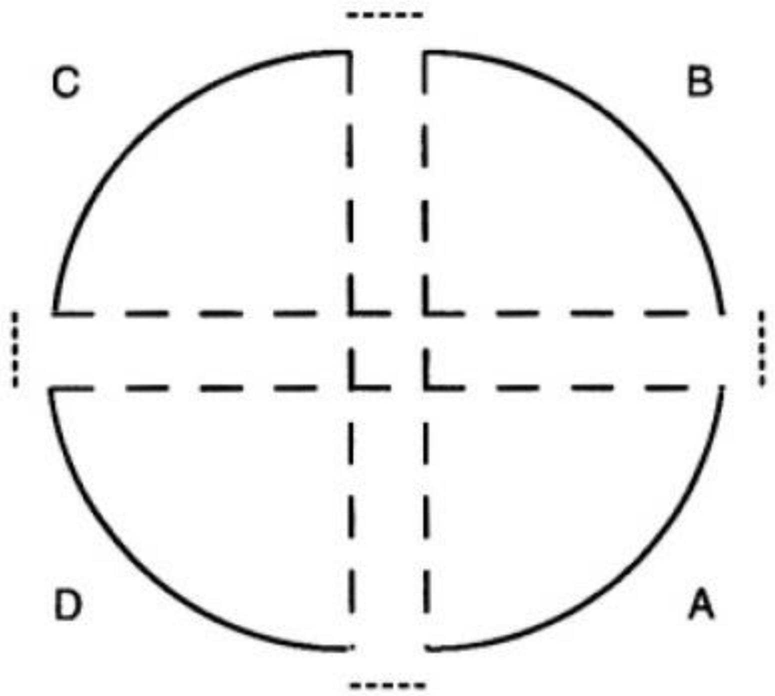
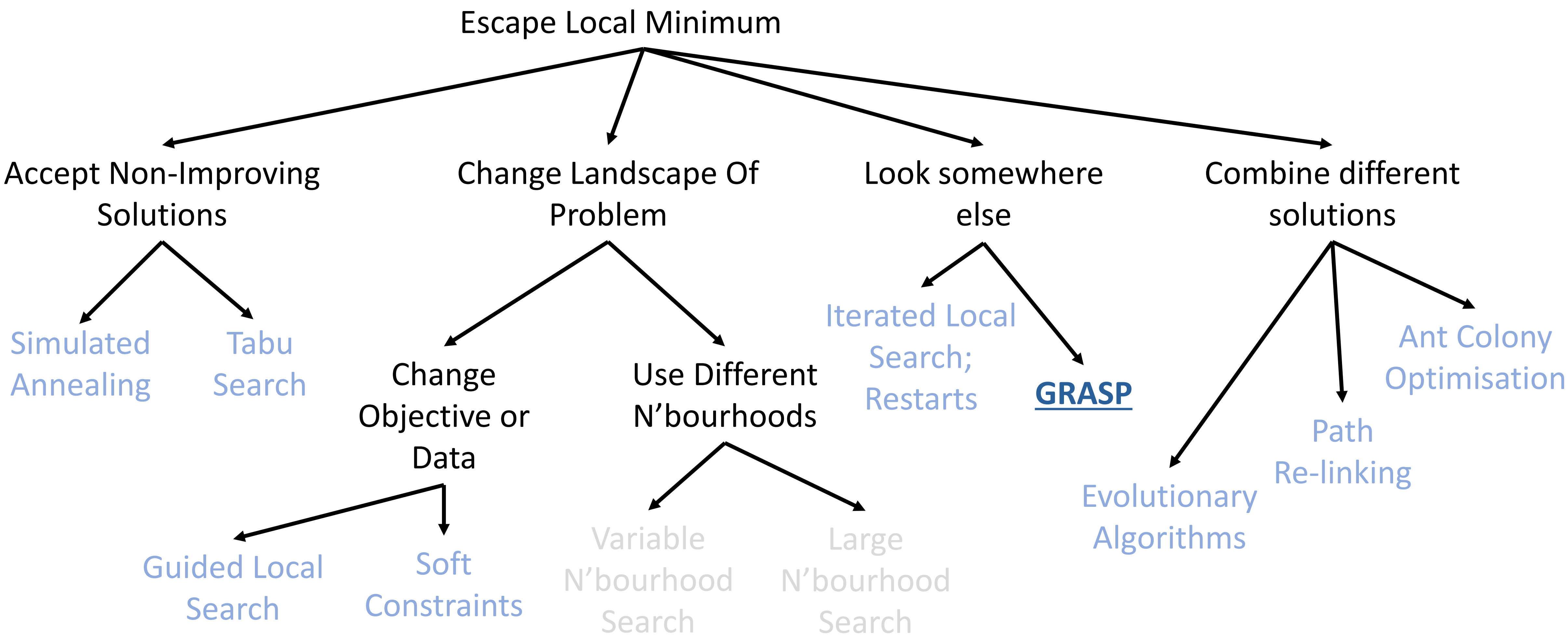


Image credit: Lourenço H.R., Martin O.C., Stützle T. (2003) Iterated Local Search

Metaheuristics



GRASP

Greedy **R**andomised **As**cent **P**rocedure

- Similar to ILS (independently developed)
- “Kick” in ILS is replaced by a constrained construction method
  - Inserts elements of the solution one at a time
  - **Guides and randomises a good order for insertion**
  - Uses a Reduced Candidate List (RCL) to guide insertion

- Based on “Feature Cost”
- E.g. TSP:
  - Feature = City
  - Feature cost = Cost of inserting city into solution (Min Insert Cost)

GRASP

For TSP:

- Feature = City
- Feature cost = Cost of inserting city into solution (Min Insert Cost)

Input: α

Soln = *empty*

repeat

```
    FeatCost_i = CostToAddFeature (Soln, i)   for i not in Soln
    minCost = min (FeatCost)
    maxCost = max (FeatCost)
    RCL = {}
    for i not in Soln
        if FeatCost_i ≤ minCost + α (maxCost – minCost)
            RCL.append (Feat_i)
    Feat = SelectRandomFeature (RCL)
    Add (Feat, Soln)
```

until IsComplete (Soln)

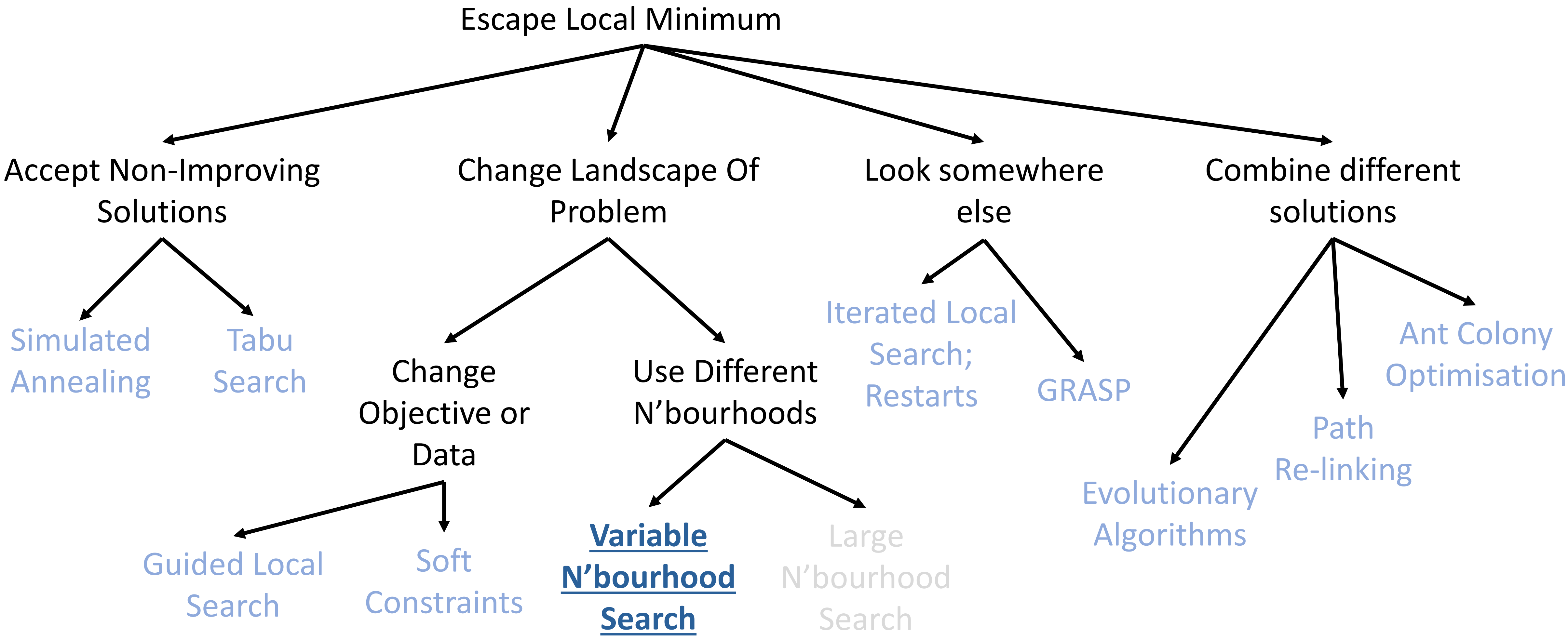
GRASP

- α too low = Greedy
- α too high = Random

- Again, methods to adapt α to problem instance

- Also, don’t have to start at empty solution every time.
  - Similar to ILS, we can destroy and reconstruct just part of the current solution

Metaheuristics



Variable Neighbourhood Search

Basic idea:

- If you have reached the local min for one neighbourhood, swap neighbourhoods (to a larger one)

Given a list of neighbourhoods $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots \mathcal{N}_m\}$

- Start by applying the first.
- When you get to local minimum, move to next
- When you find an improvement, go back to first.

Very successful method across multiple domains

Variable Neighbourhood Search

```
S = initialSolution()
k = 1
while k < m
    if LocalSearch(S, N_k) improved the solution S:
        k = 1
    else
        k = k + 1
```

Variable Neighbourhood Search

E.g. neighbouhoods for VRP

1) 1-move (move 1 visit to another position)
2) 1-1 swap (swap visits in 2 routes)
3) 2-2 swap (swap 2 visits between 2 routes)
4) Or-opt size 2 (move chain of length 2 – forwards and backwards – anywhere)
5) Or-opt size 3 (chain length 3)
6) Tail exchange (swap final portion of routes)
7) 2-opt
8) 3-opt

Variable Neighbourhood Search

VNS + ILS

- Take the “kick” idea from Iterated Local Search
  - when a local minimum is found, apply a series of neighbourhood moves *regardless of cost*

- Number to apply (strength of kick) has to be chosen carefully, as per ILS

Metaheuristics

- Escape Local Minimum
  - Accept Non-Improving Solutions
    - Simulated Annealing
    - Tabu Search
  - Change Landscape Of Problem
    - Change Objective or Data
      - Guided Local Search
      - Soft Constraints
    - Use Different N'bourhoods
      - Variable N'bourhood Search
      - **Large N'bourhood Search**
  - Look somewhere else
    - Iterated Local Search; Restarts
    - GRASP
  - Combine different solutions
    - Evolutionary Algorithms
    - Path Re-linking
    - Ant Colony Optimisation

Large Neighbourhood Search

- Originally developed by Paul Shaw (1997)
- This version Ropke & Pisinger (2007)[1]
- A.k.a "Record-to-record" search

VRP as an example:
- Remove customers
- Re-insert those customers

- Destroy part of the solution
  - Remove elements from the solution
- Re-create solution
  - Use favourite construct method to re-insert those elements
- If the solution is better, keep it
- Repeat

1: S Ropke and D Pisinger, *An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows*, Transportation Science **40**(4), pp 455-472, 2007

Large Neighbourhood Search

Destroy part of the solution (Select method)

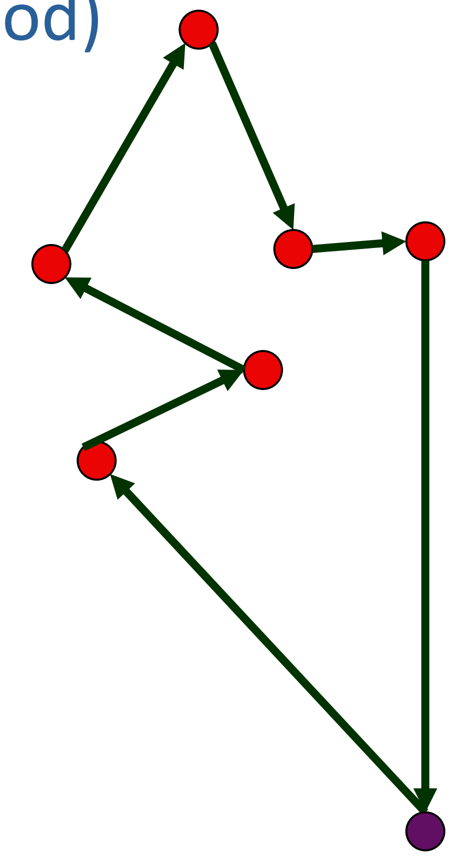
In VRP:

- Remove some visits
- Move them to the "unassigned" lists

Large Neighbourhood Search

Destroy part of the solution (Select method)

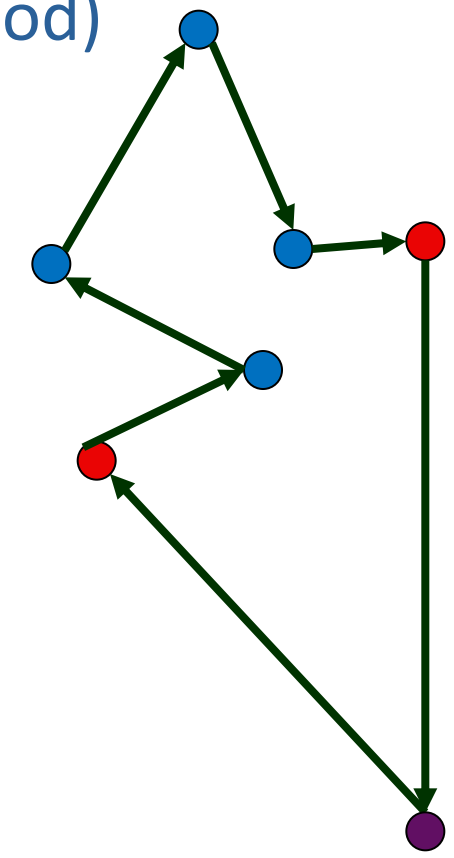
- Examples
- Remove a sequence of visits



Large Neighbourhood Search

Destroy part of the solution (Select method)

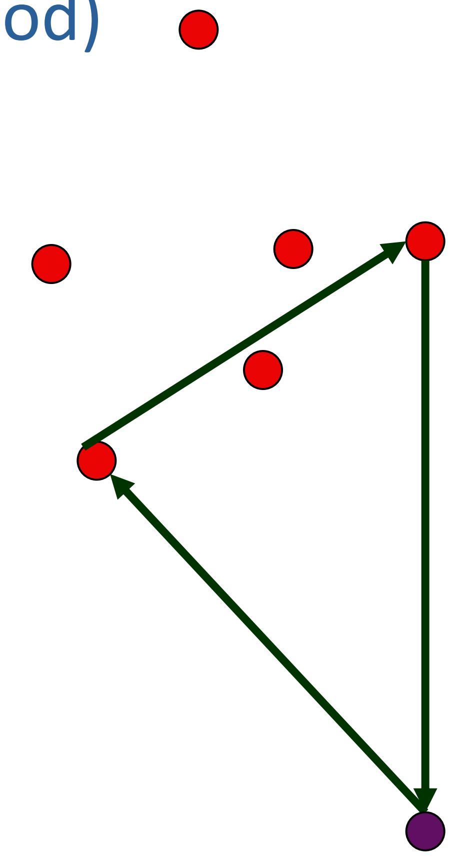
- Examples
- Remove a sequence of visits



Large Neighbourhood Search

Destroy part of the solution (Select method)

- Examples
- Remove a sequence of visits

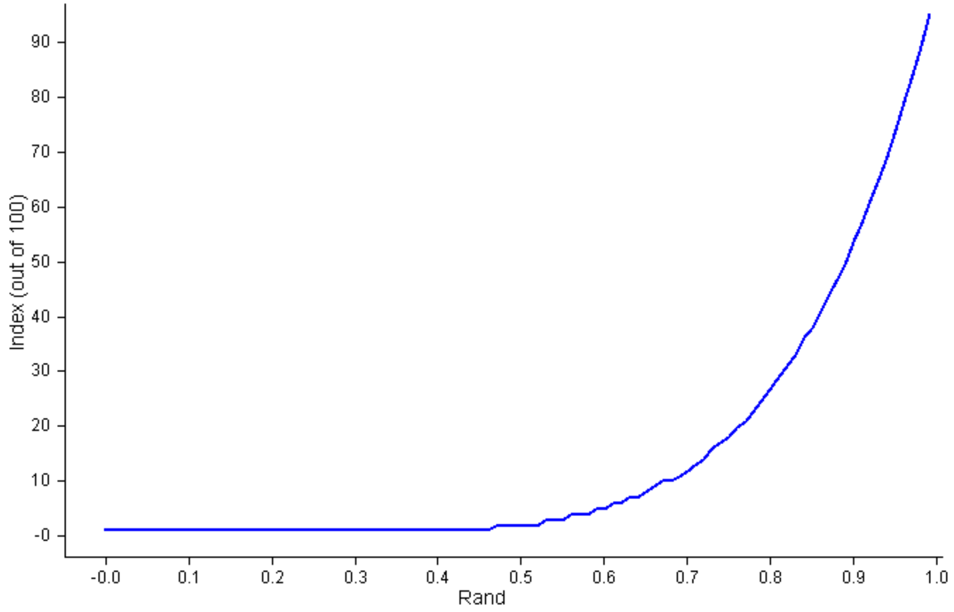


Large Neighbourhood Search

Destroy part of the solution (Select method)

Examples

- Choose longest (worst) arc in solution
  - Remove visits at each end
  - Remove nearby visits
- Actually, choose `r`th worst
- `r = n` * (uniform(0,1))^y
- `y` ~ 6
  - unif → (unif)^y
  - 0.56 → 0.016
  - 0.96 → 0.531

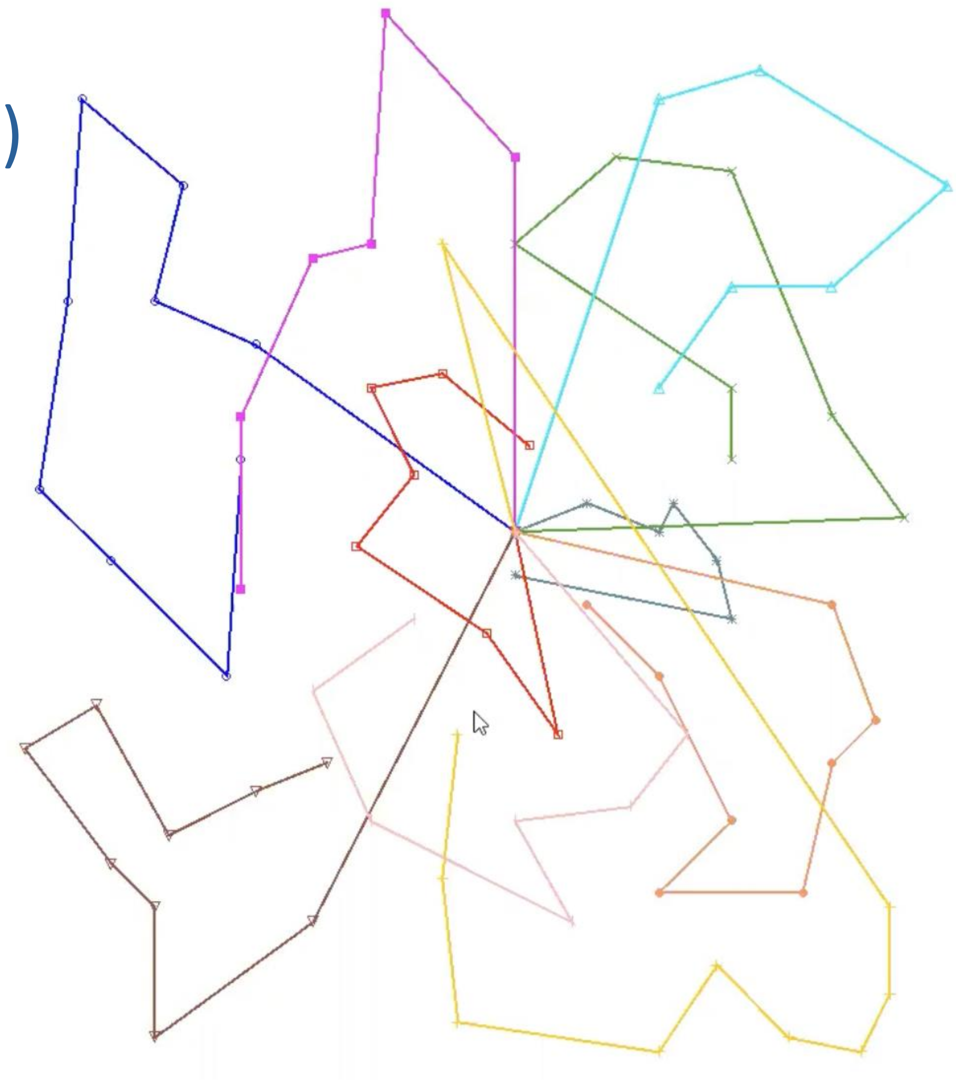


Large Neighbourhood Search

Destroy part of the solution (Select method)

Examples

- Dump all visits from *k* routes (k = 1, 2, 3)
  - Prefer routes that are close,
  - Better yet, overlapping



Large Neighbourhood Search

Destroy part of the solution (Select method)

Examples

- Choose first visit randomly
- Then, remove "related" visits
  - Based on distance, time compatibility, load

$$R_{ij} = \varphi C_{ij} + \quad \text{Load}$$

$$\chi(|a_i - a_j|) + \quad \text{Distance}$$

$$\psi(|q_i - q_j|) \quad \text{Time}$$

Large Neighbourhood Search

Destroy part of the solution (Select method)

Examples

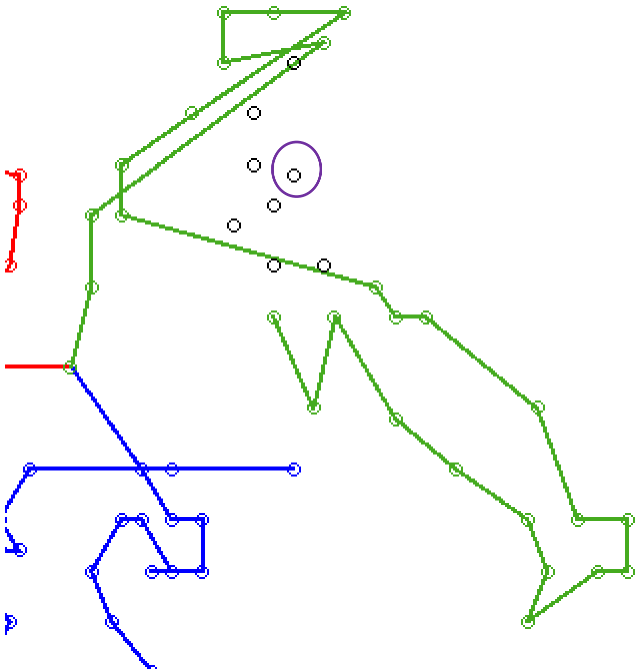
- Dump visits from the smallest route
  - Good if saving vehicles
  - Sometimes fewer vehicles = reduced travel

Large Neighbourhood Search

Destroy part of the solution (Select method)

Examples

- Nearest neighbour
  - Select first customer randomly
  - Select *n* nearest neighbours
    - Allows us to find a better tour in a local area



Large Neighbourhood Search

Destroy part of the solution (Select method)

- Parameter: Max to dump
  - As a % of n?
  - As a fixed number e.g. 100 for large problems
- Actual number is uniform rand (5, max)

Large Neighbourhood Search

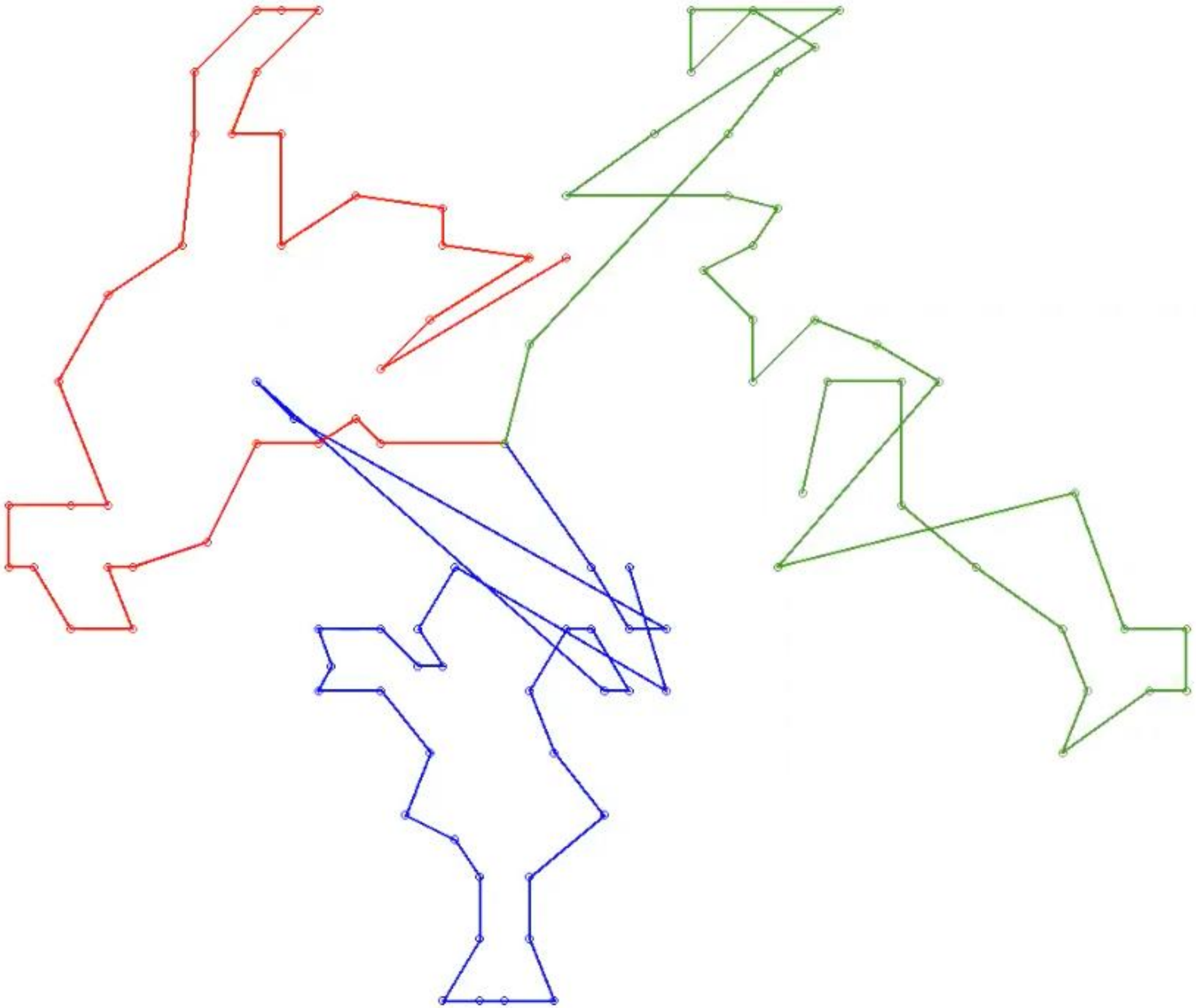
Re-create solution

- Systematic search (e.g., MILP, Constrained Programming, etc)
  - Smaller problem, easier to solve
  - **Can be very effective**
- Use your favourite insert method
- Better still, use a portfolio of insert methods
  - Ropke's paper[1]: Select amongst
    - Minimum Insert Cost
    - Regret (2-regret)
    - 3-regret
    - 4-regret
    - Random insert order

1: S Ropke and D Pisinger, *An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows*, Transportation Science **40**(4), pp 455-472, 2007

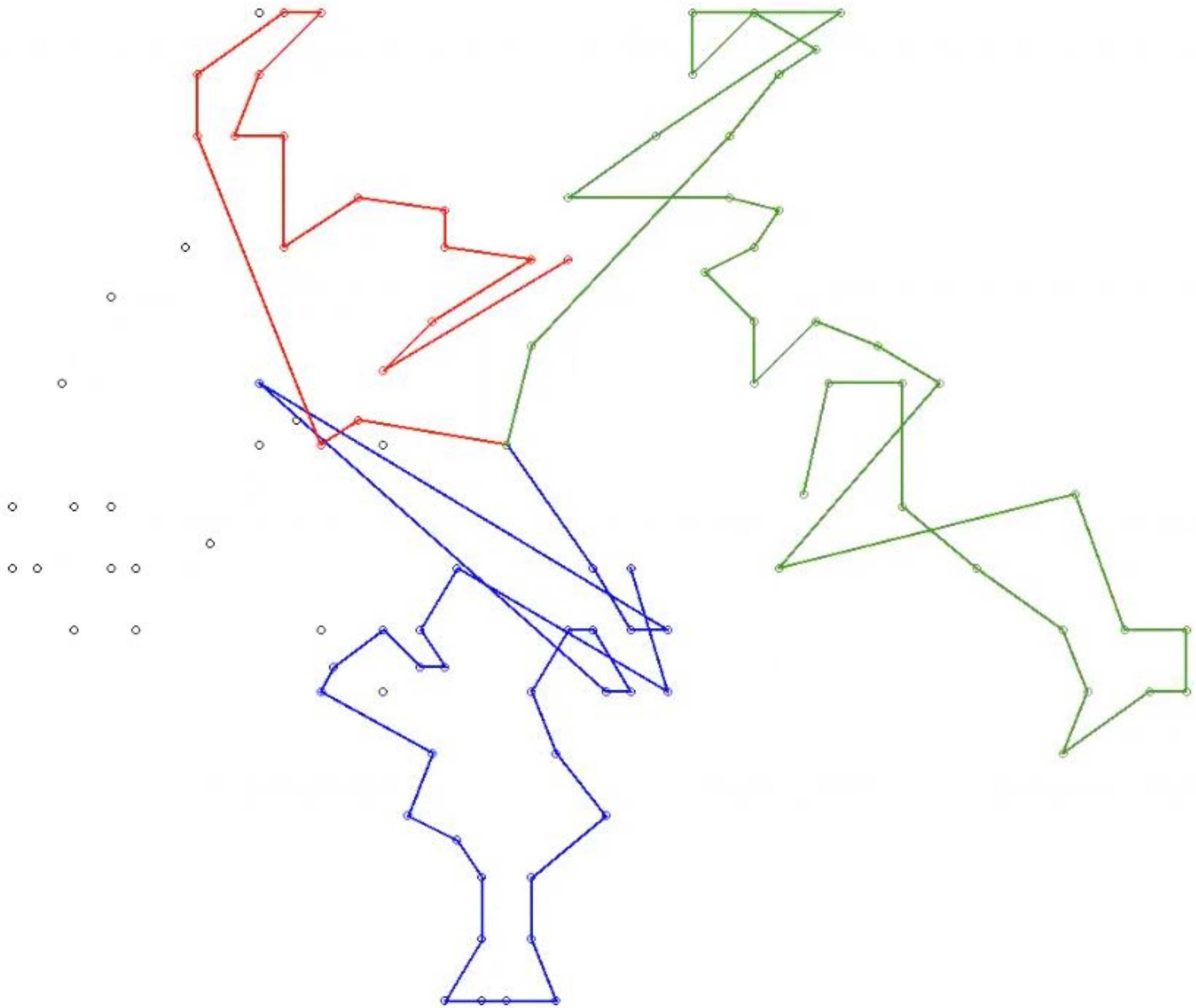
Large Neighbourhood Search – VRP

Initial solution



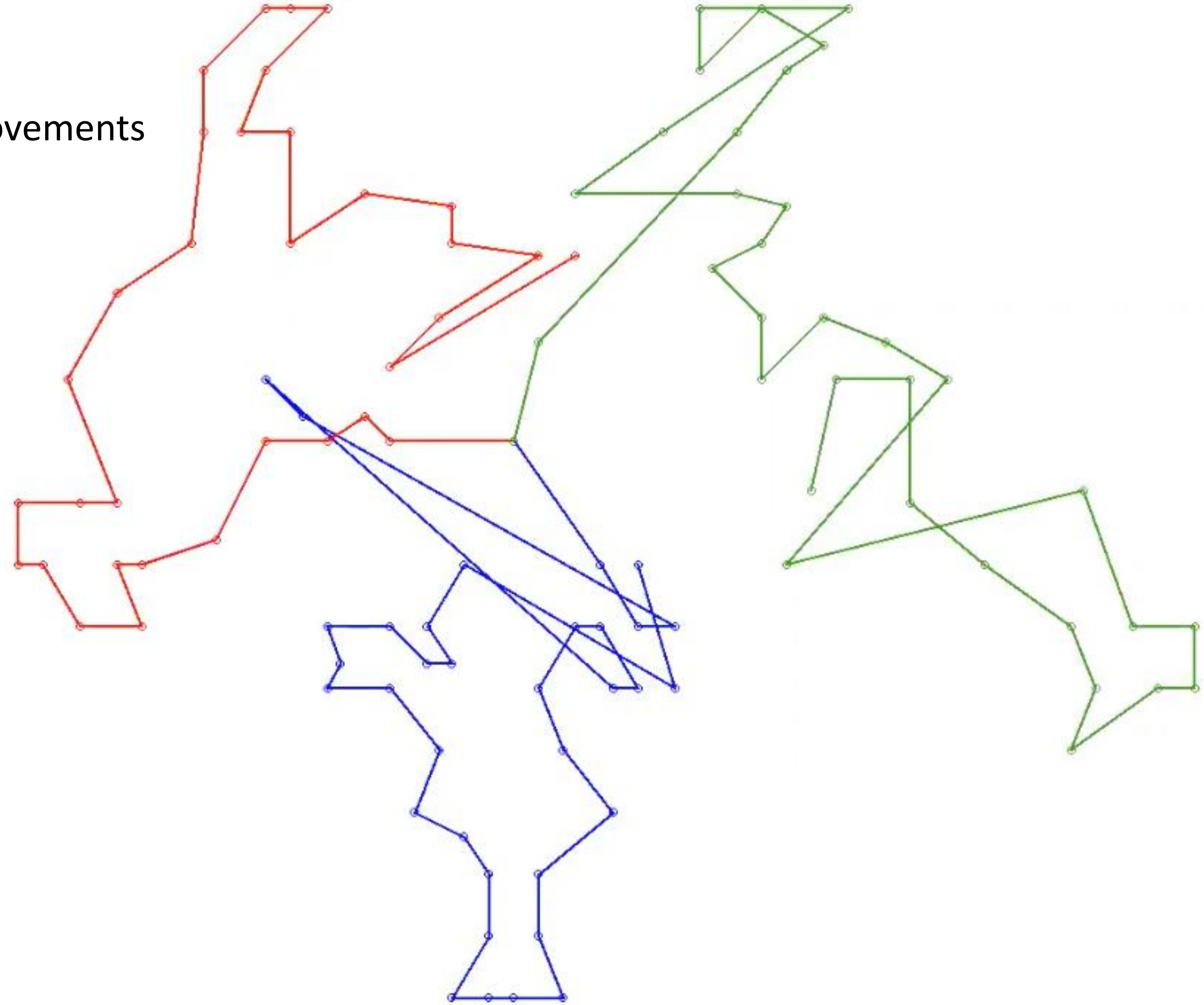
Large Neighbourhood Search – VRP

Destroy using relatedness



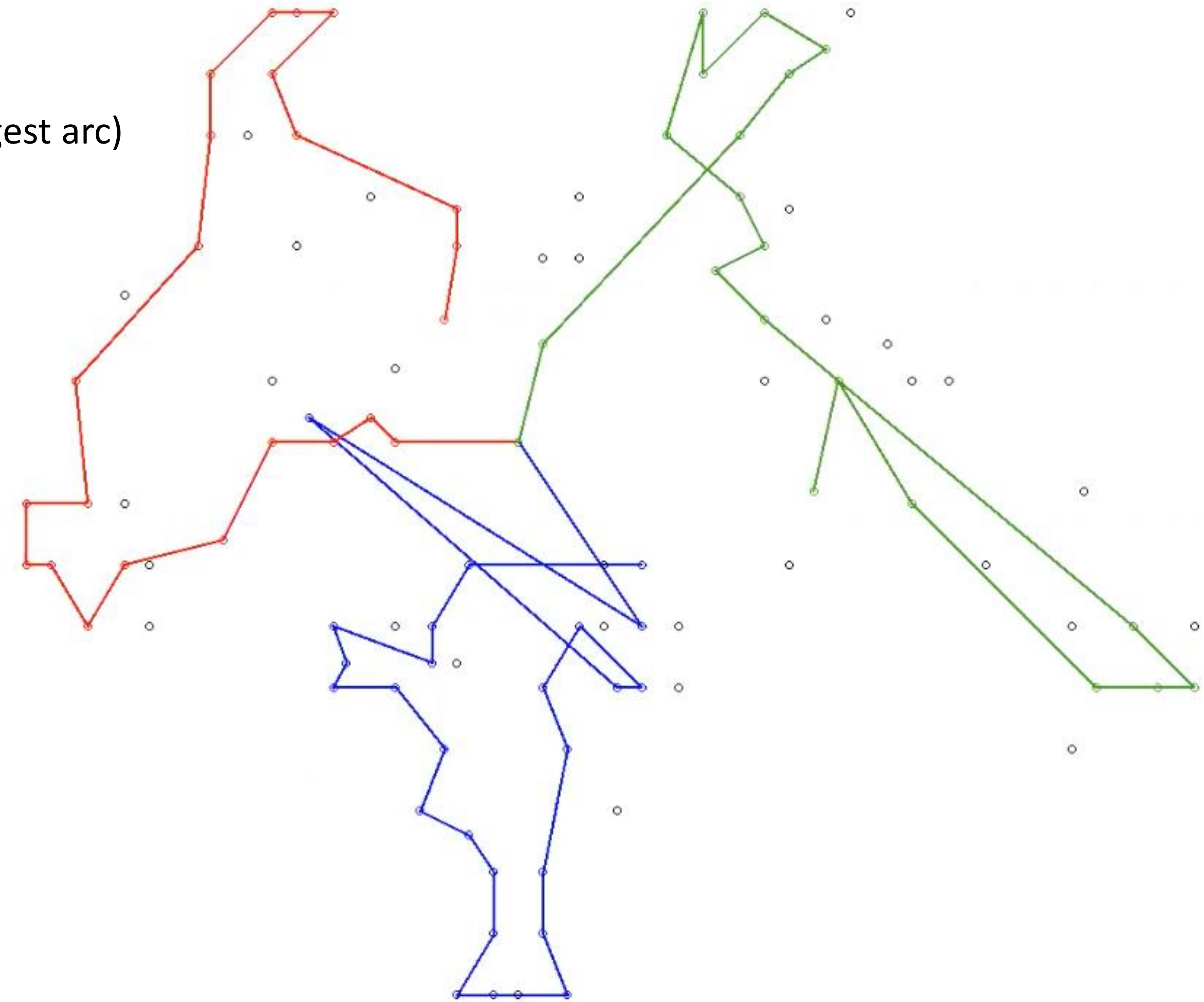
Large Neighbourhood Search – VRP

Reconstruct and no improvements



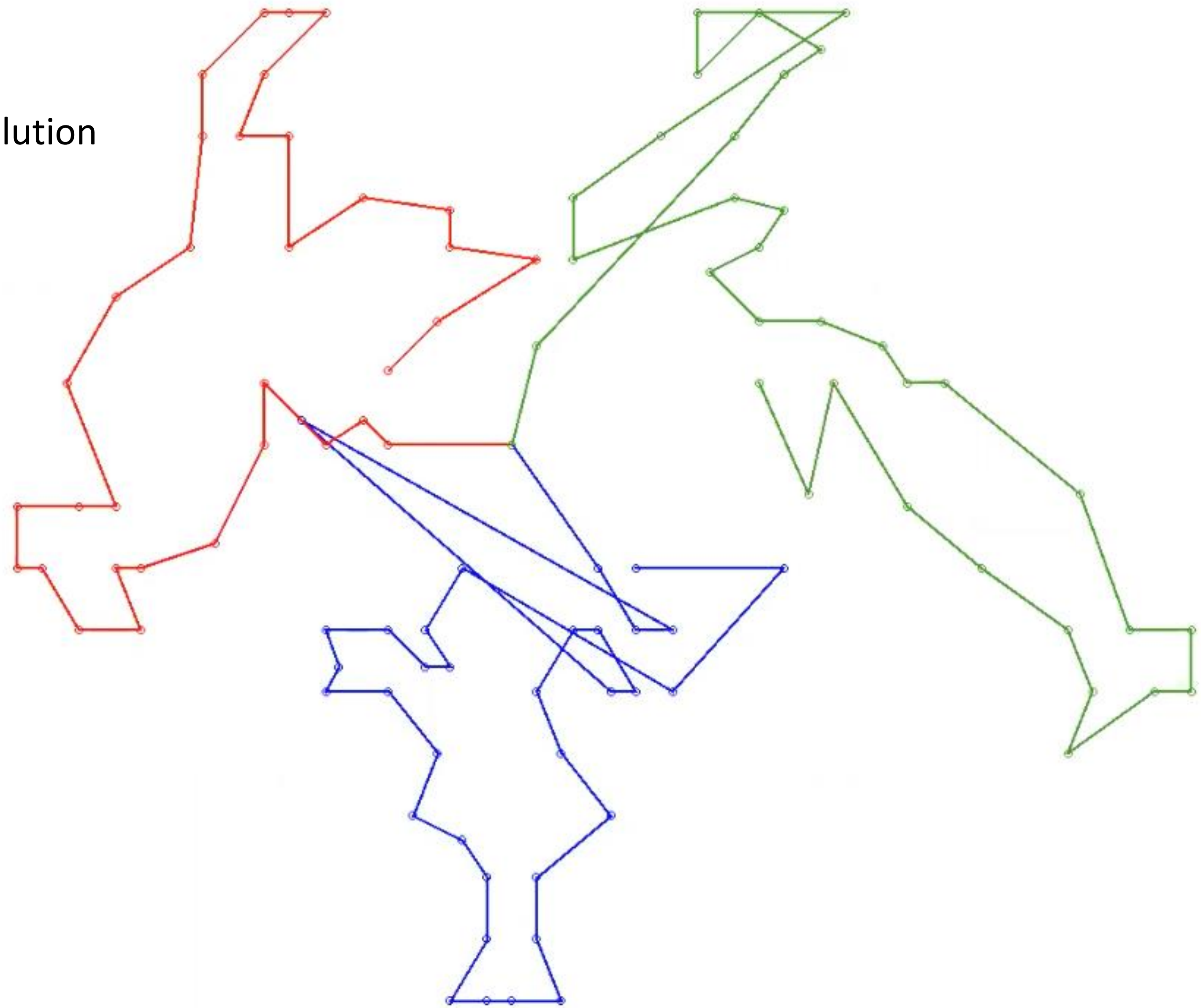
Large Neighbourhood Search – VRP

Destroy using worst (longest arc)



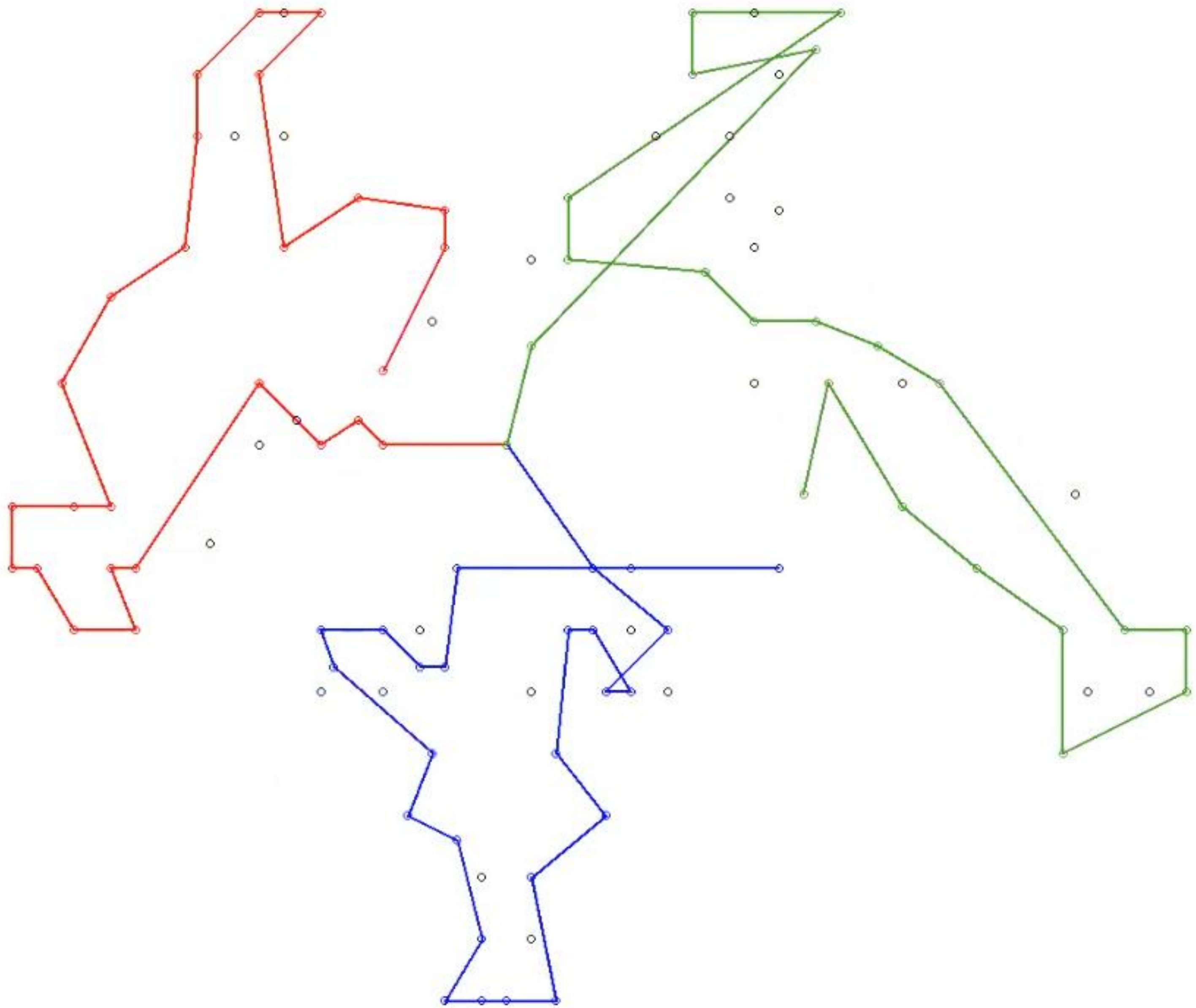
Large Neighbourhood Search – VRP

Reconstruct and better solution found



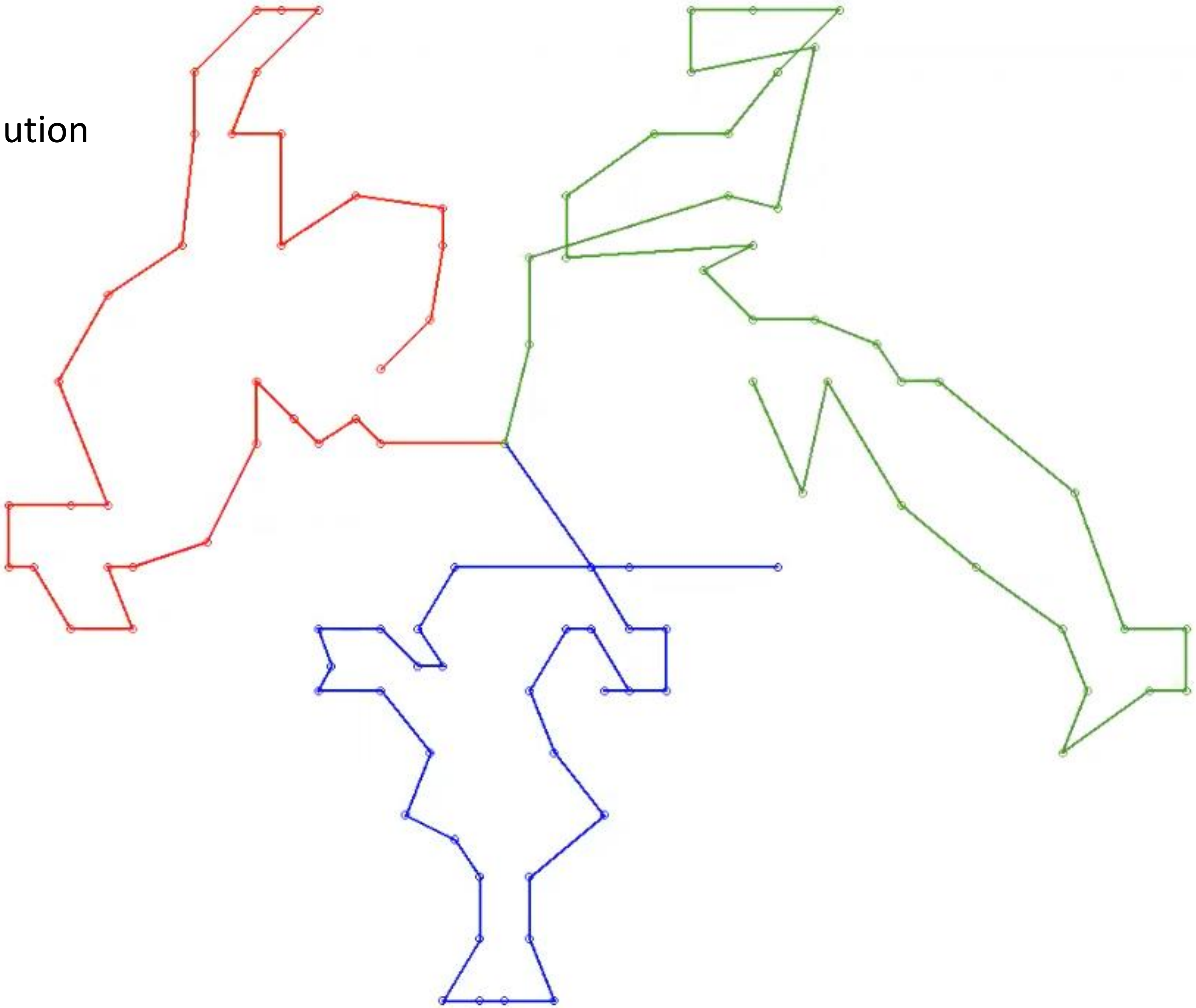
Large Neighbourhood Search – VRP

Destroy using random



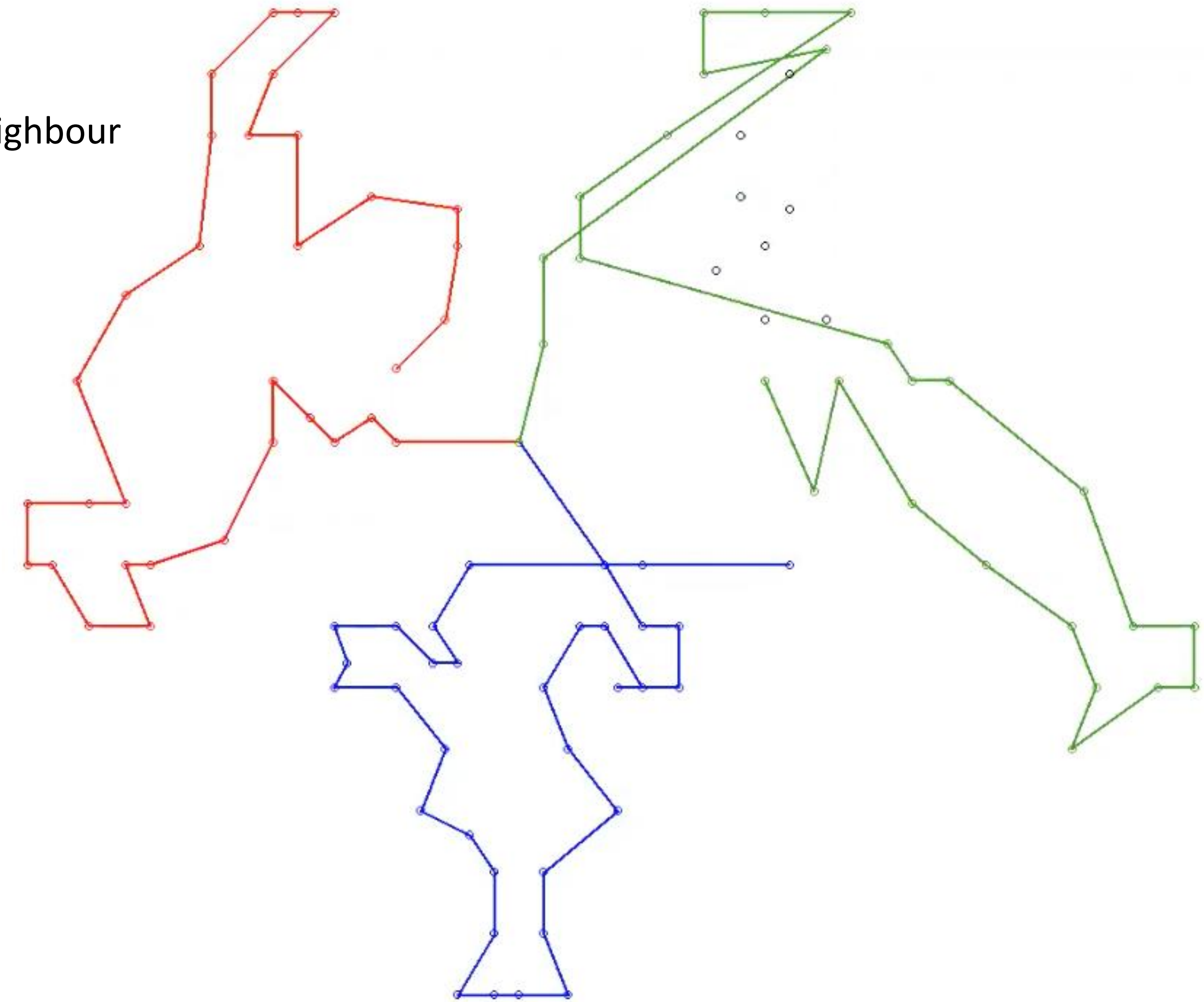
Large Neighbourhood Search – VRP

Reconstruct and better solution found



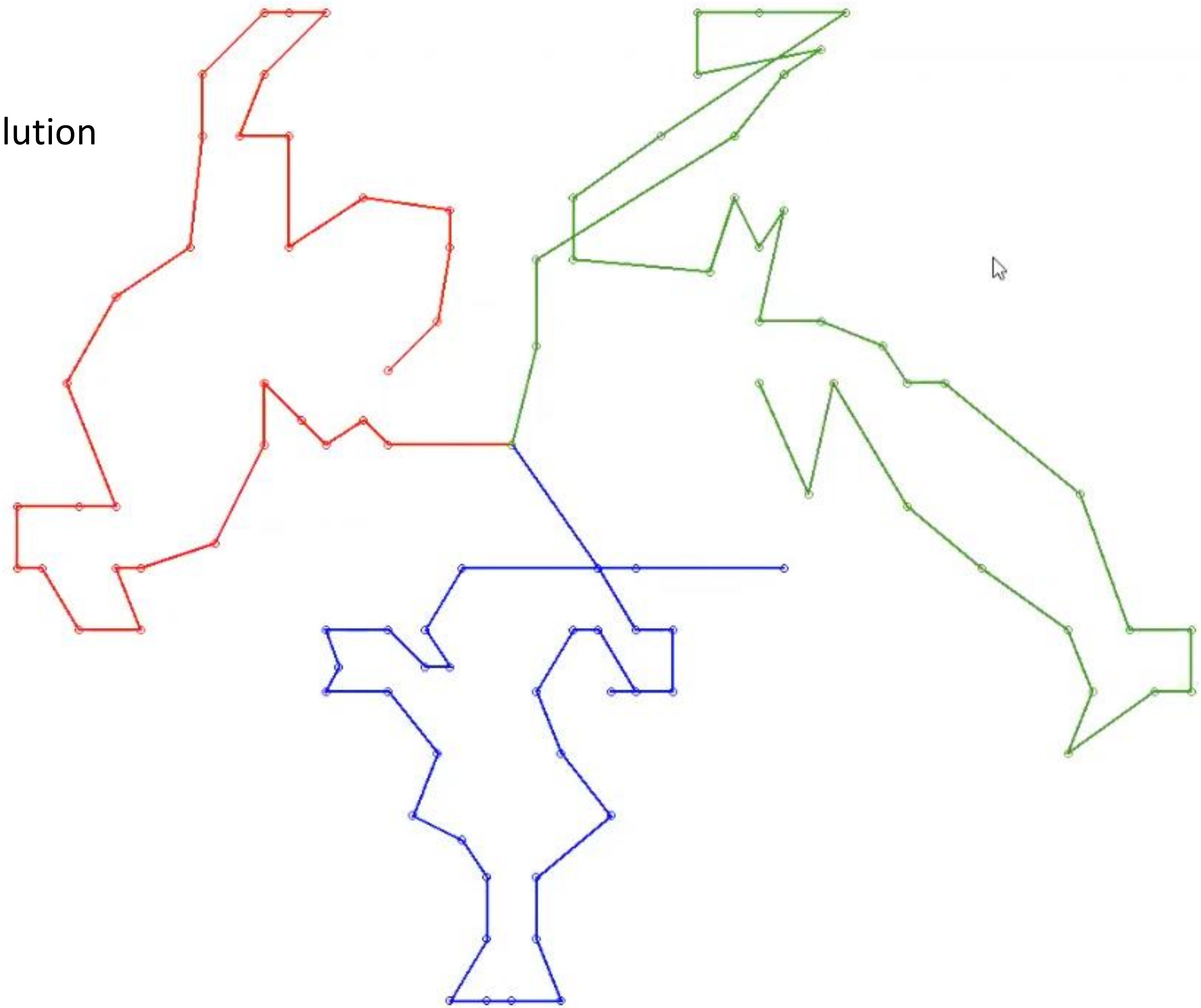
Large Neighbourhood Search – VRP

Destroy using nearest neighbour



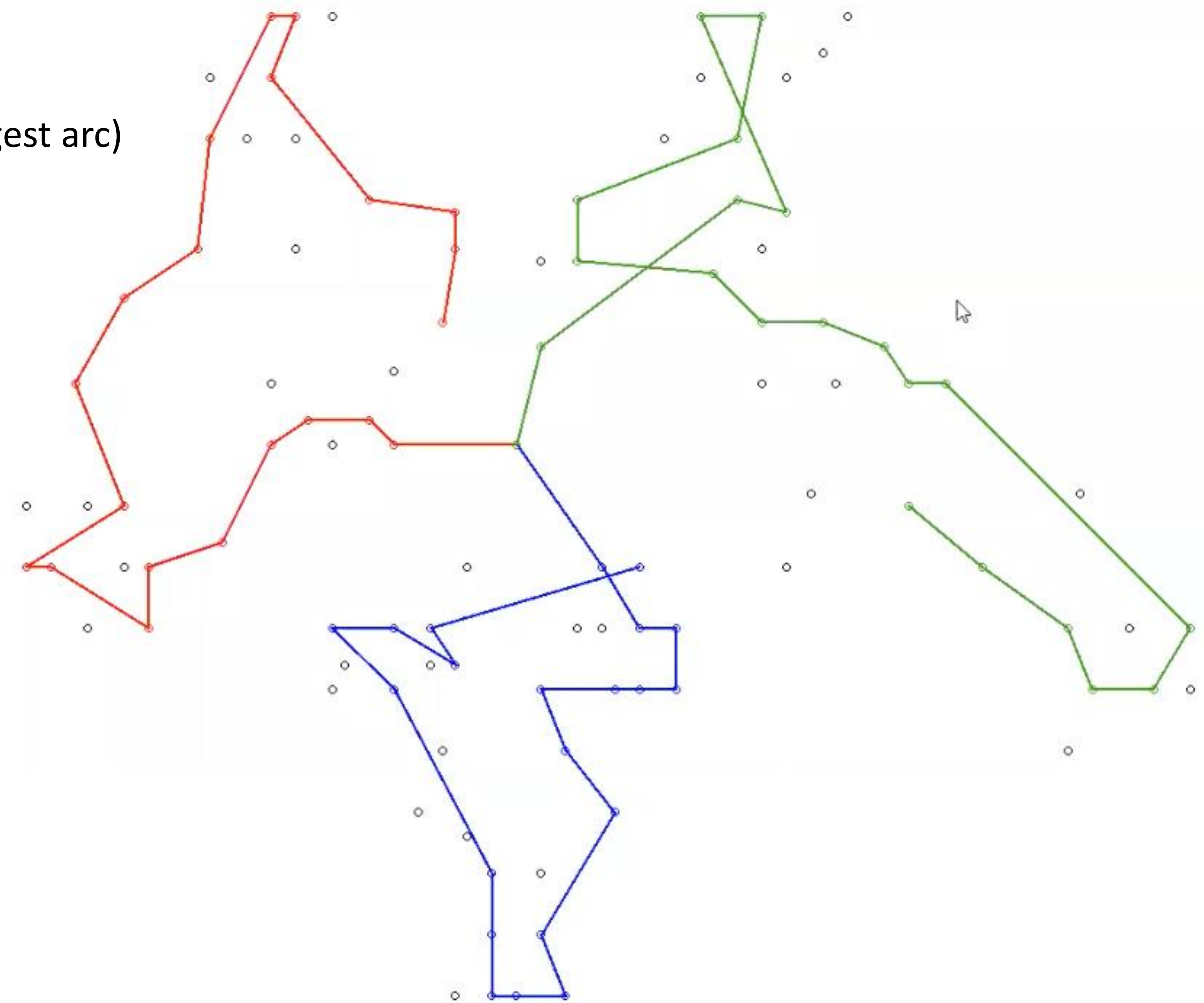
Large Neighbourhood Search – VRP

Reconstruct and better solution found



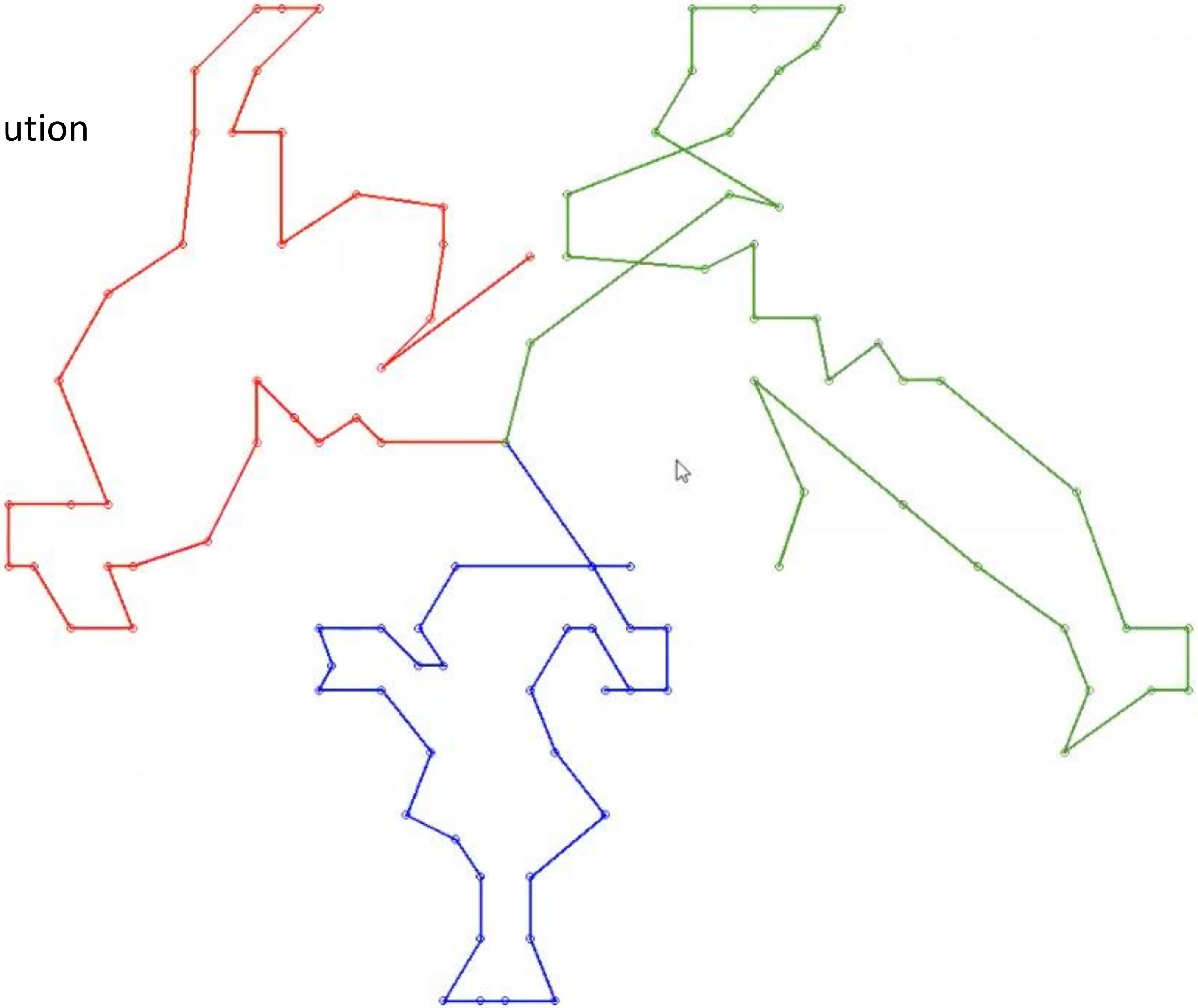
Large Neighbourhood Search – VRP

Destroy using worst (longest arc)



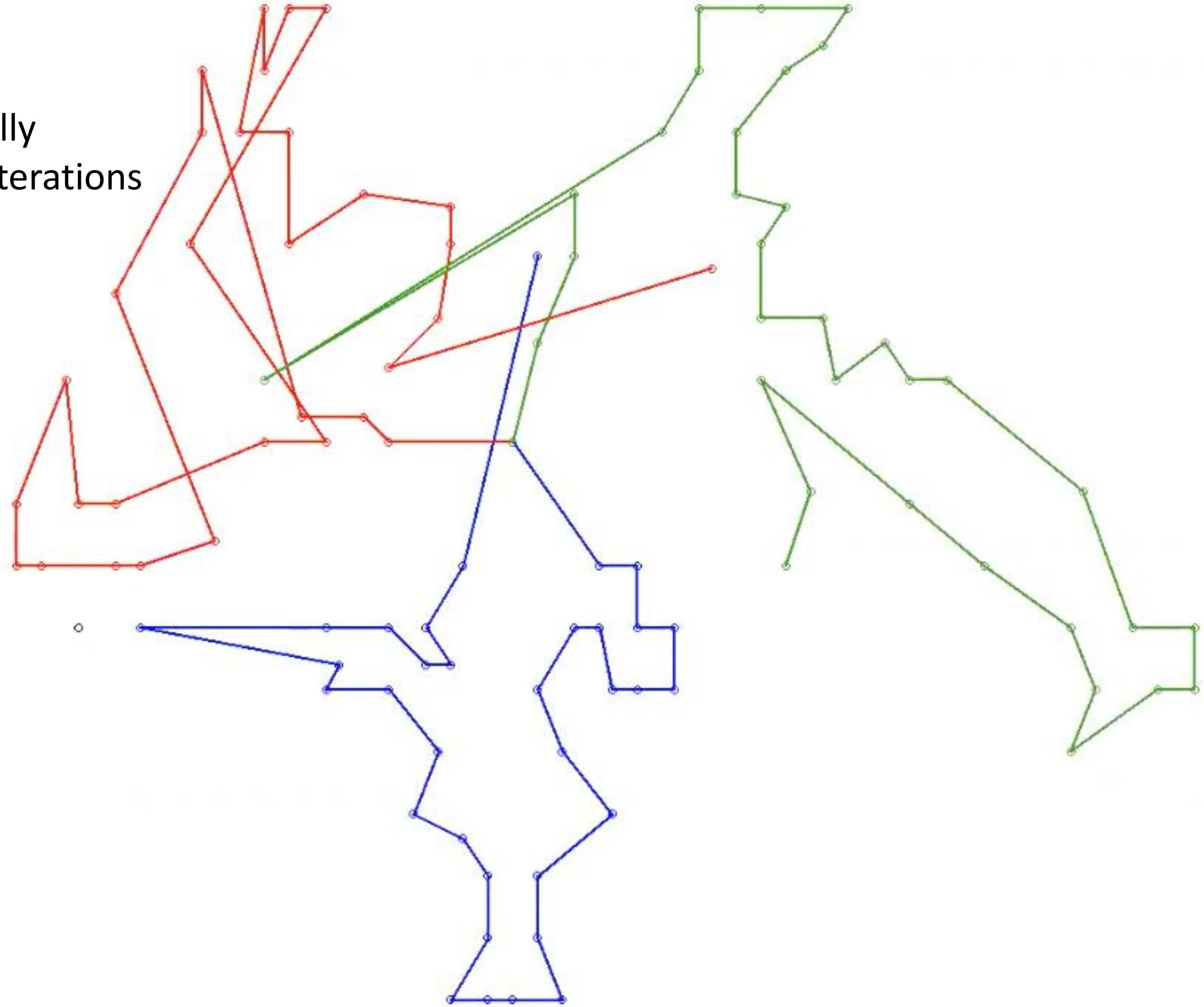
Large Neighbourhood Search – VRP

Reconstruct and better solution found



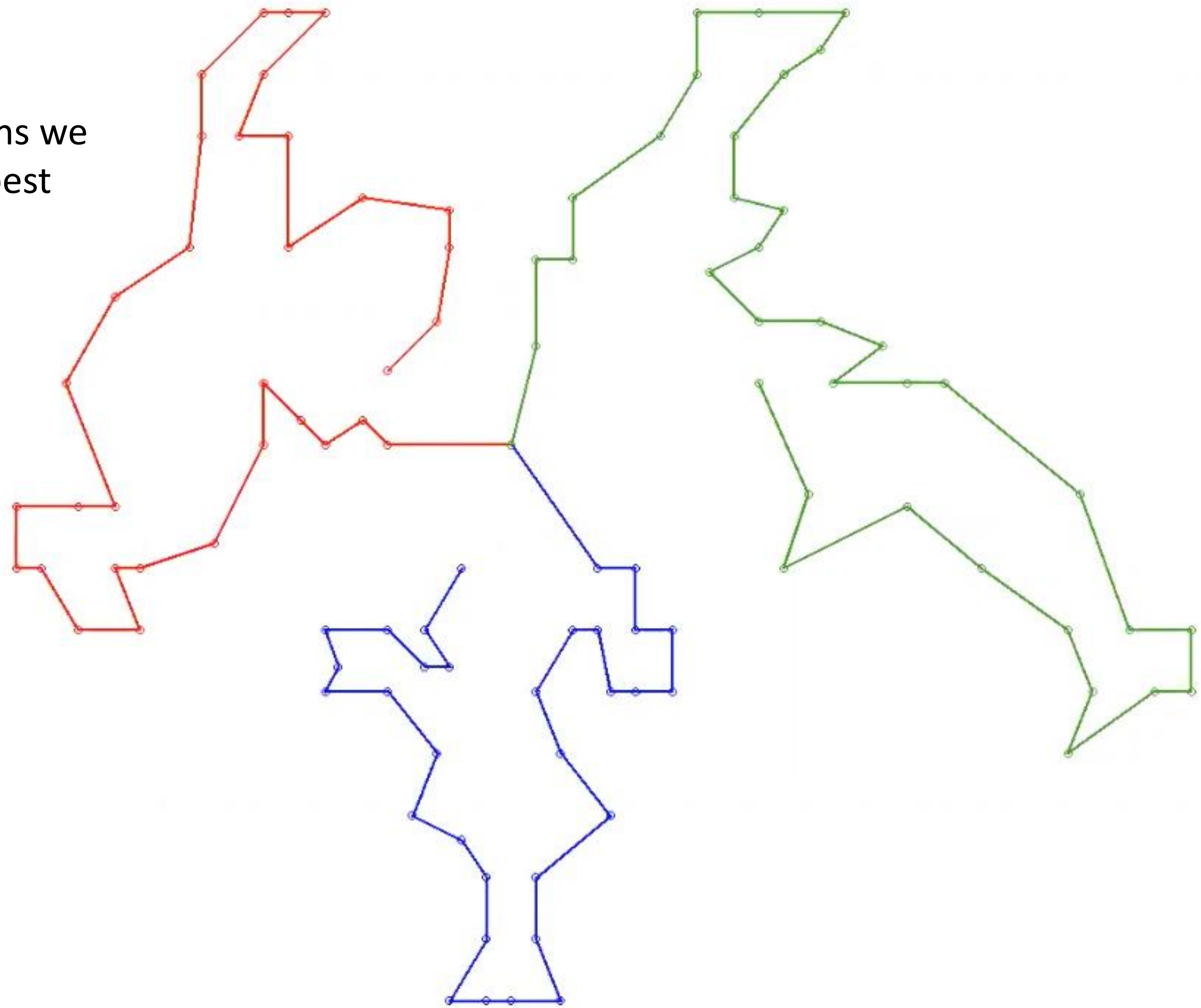
Large Neighbourhood Search – VRP

We can observe some really bad solutions during the iterations



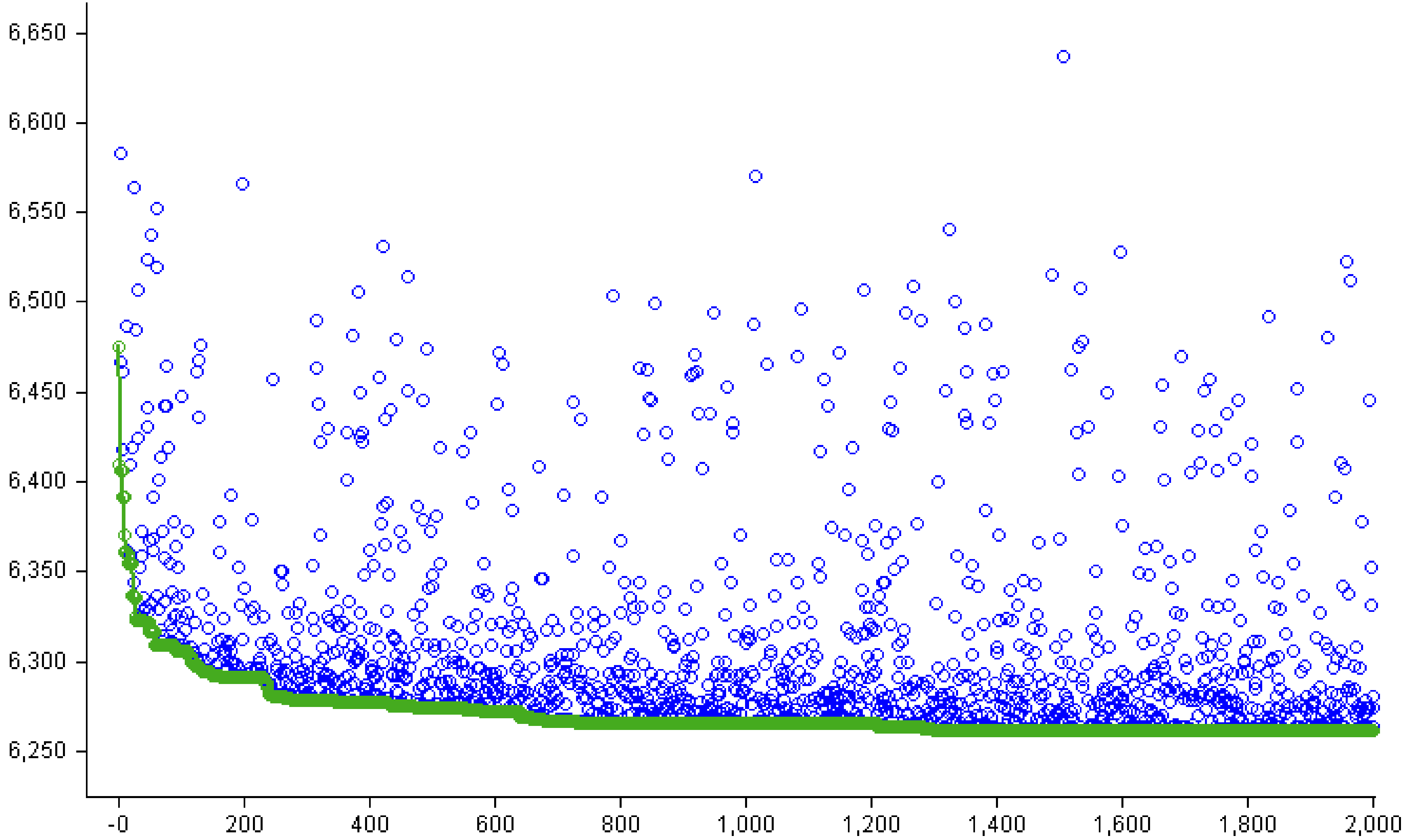
Large Neighbourhood Search – VRP

After less than 50 iterations we get this route that is the best known for this problem



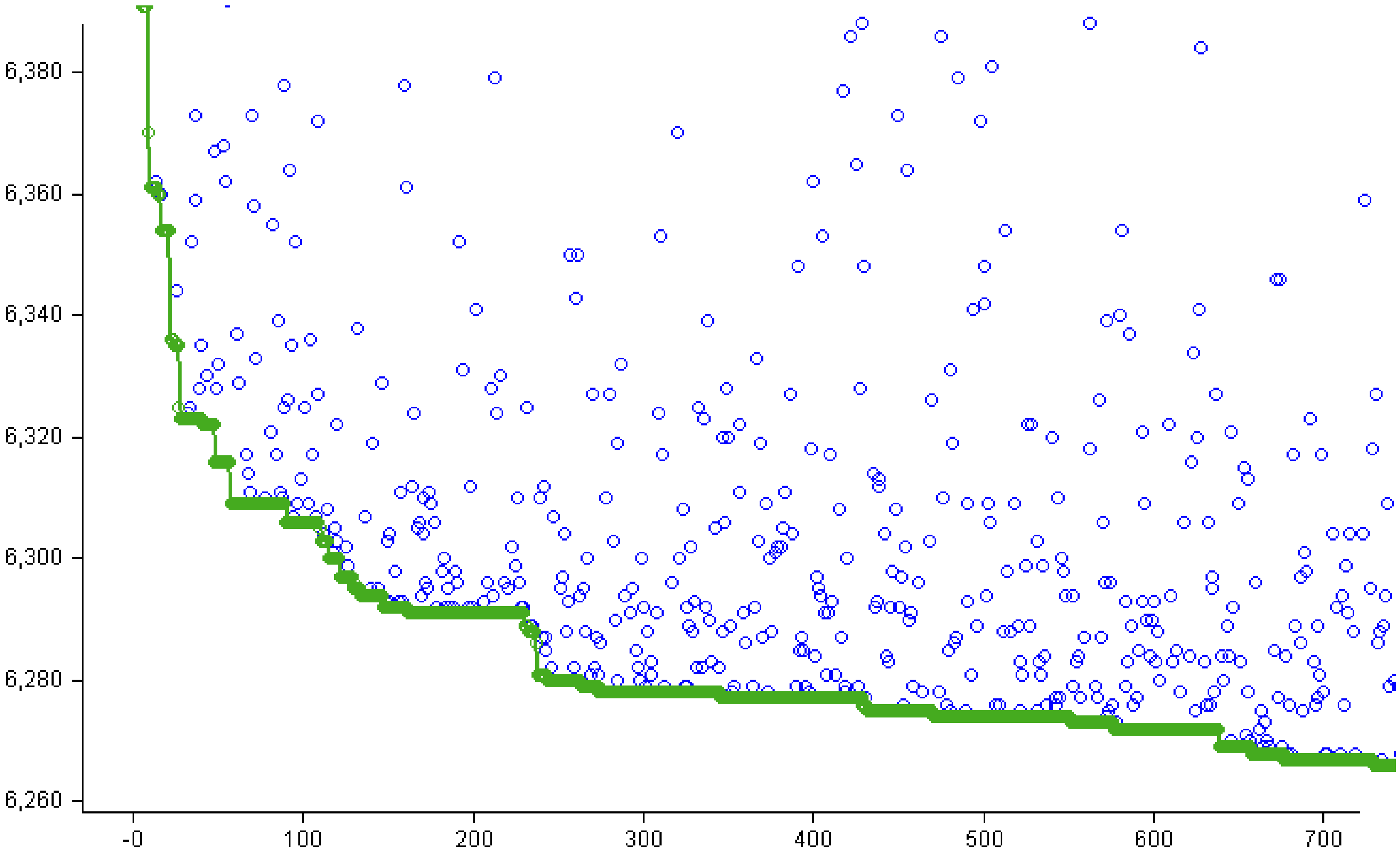
Large Neighbourhood Search

- If the solution is better, keep it



Large Neighbourhood Search

- If the solution is better, keep it



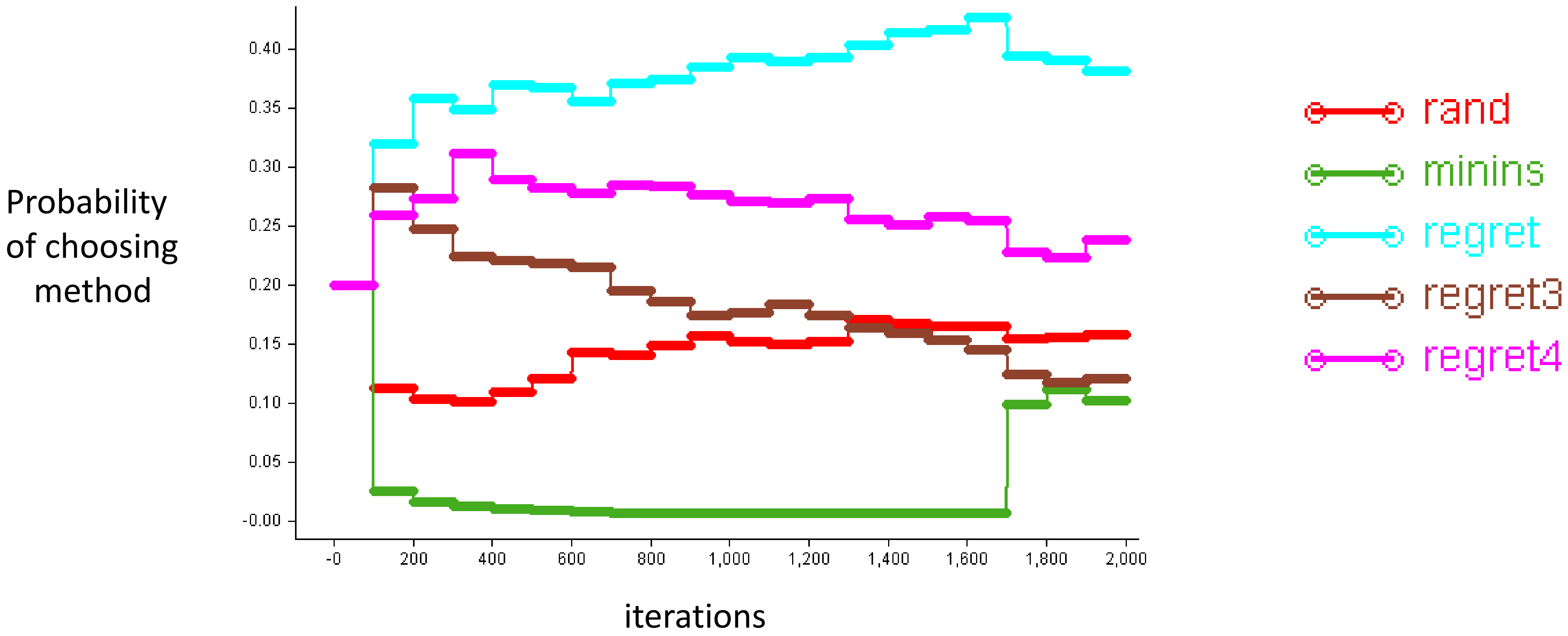
Large Neighbourhood Search

- If the solution is better, keep it
- Can use Hill-climbing
- Can use Simulated Annealing
- …

Large Neighbourhood Search

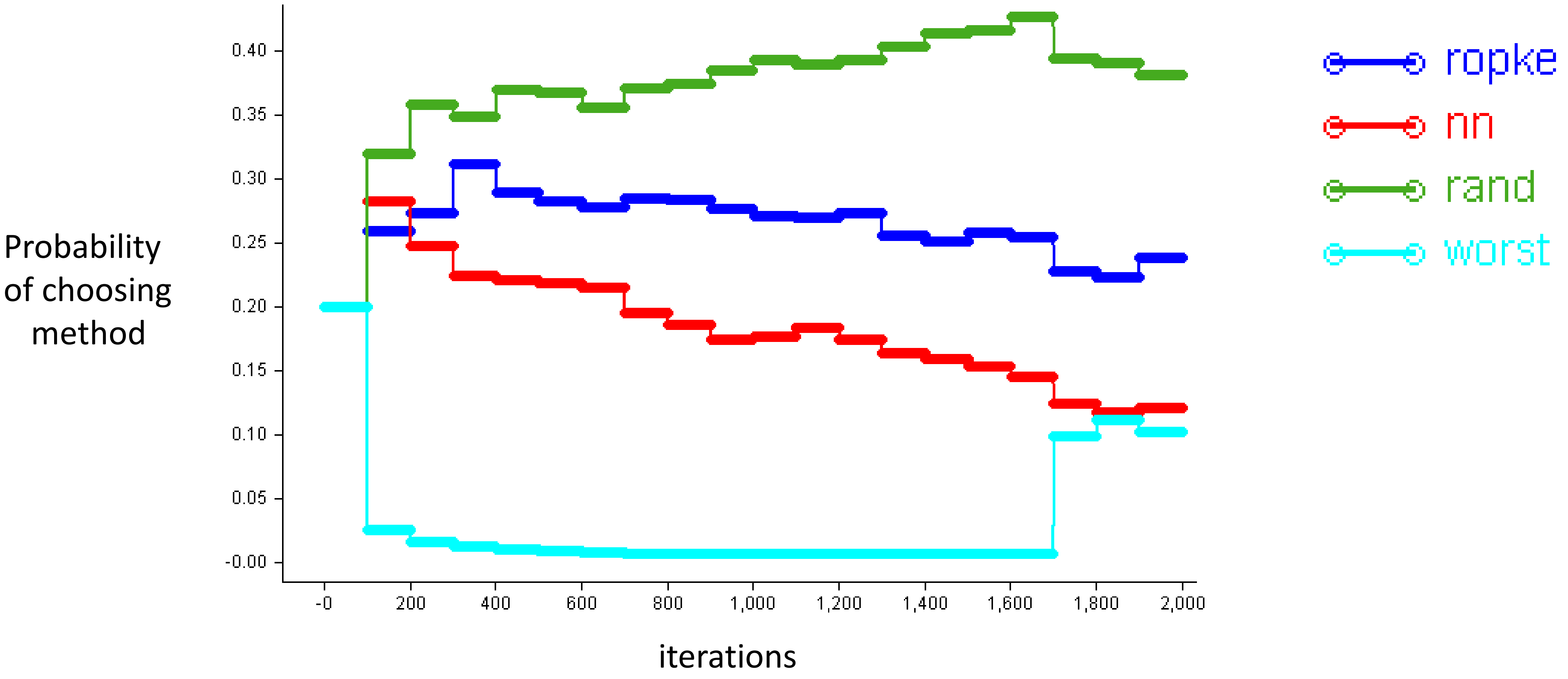
Adaptive Insert Method

- Ropke[1] adapts choice based on prior performance
  - “Good” methods are chosen more often



Large Neighbourhood Search

Adaptive Select Method (destruction method)



Large Neighbourhood Search

Ropke & Pisinger (with additions) can solve a variety of problems

- VRP
- VRP + Time Windows
- Pickup and Delivery
- Multiple Depots
- Multiple Commodities
- Heterogeneous Fleet
- Compatibility Constraints

Large Neighbourhood Search

More generally:

As soon as you have a construction method you trust,
you have an LNS

- (Random, clustered, and various other Select methods are easy to define)

- That makes it a very powerful idea

Meta-heuristics: An Incomplete Survey