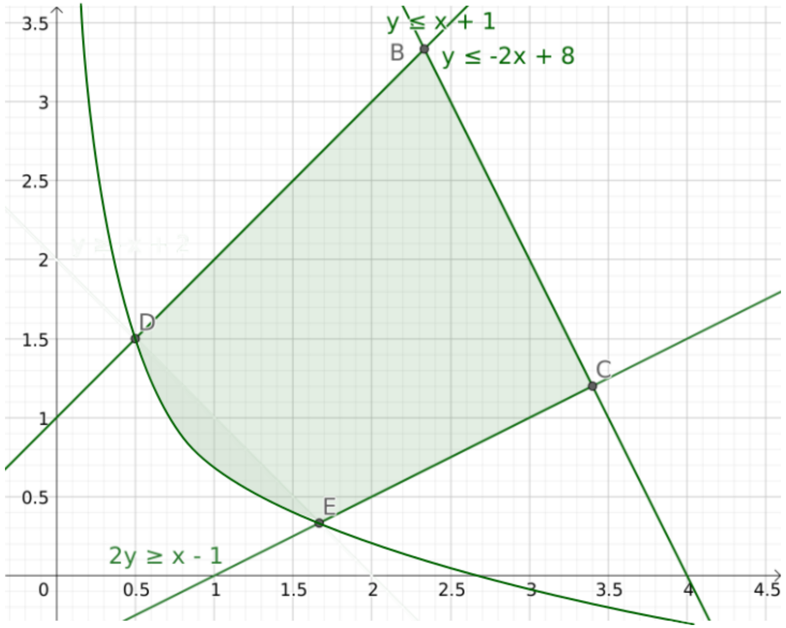
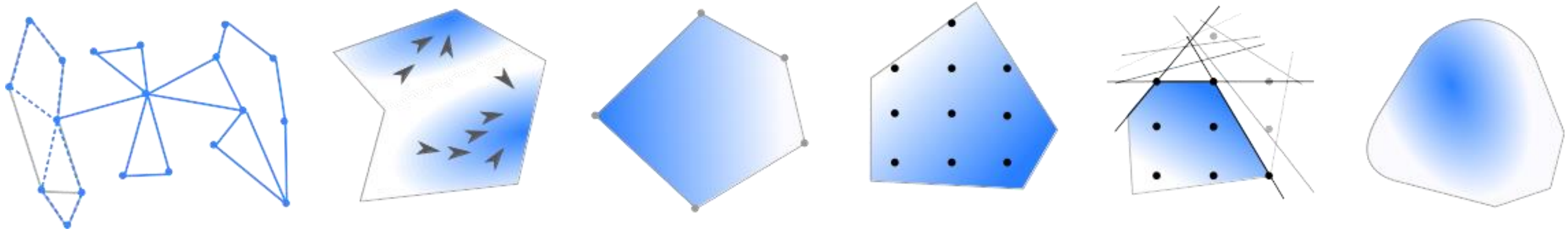
Meta-Heuristics 2

COMP4691 / 8691





Previously on COMP4691(8691)

CONSTR.

3 C

LAST MH

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

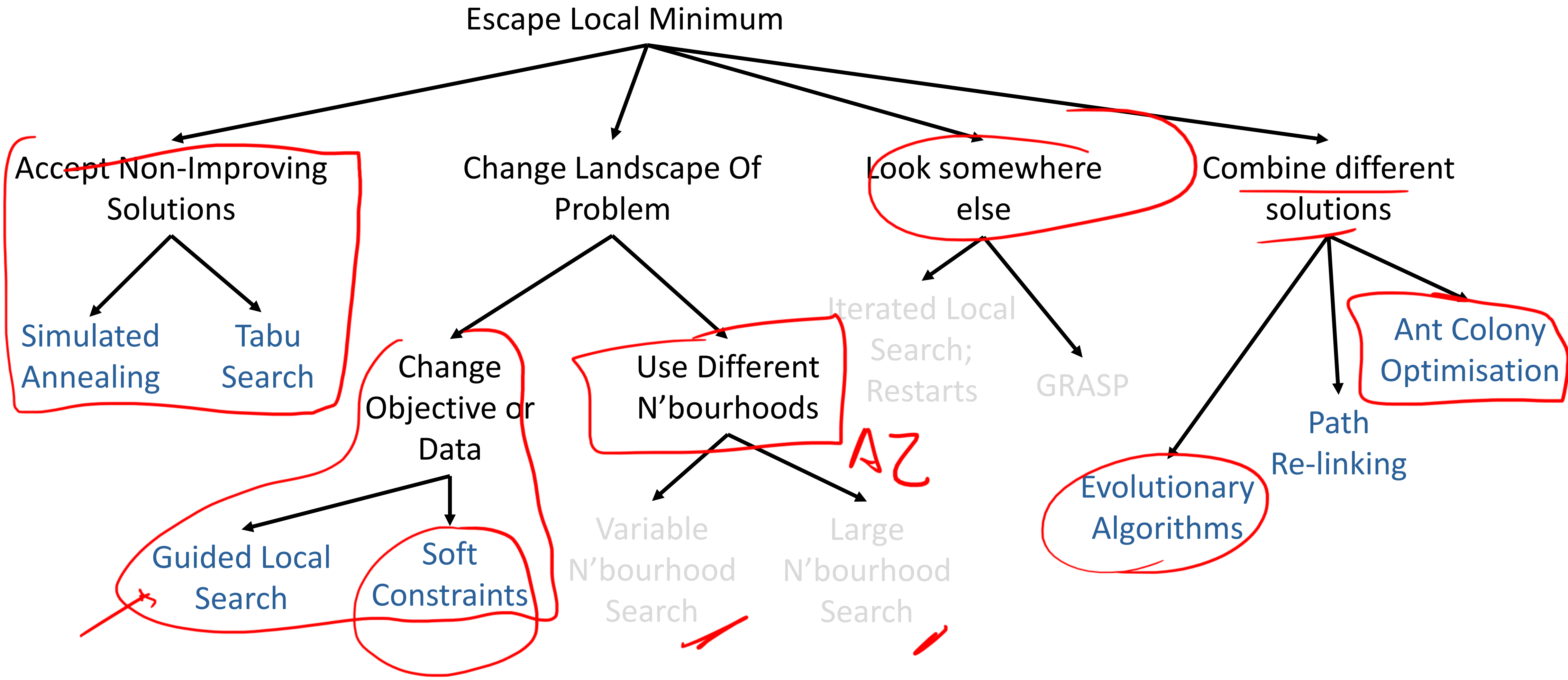
- Some Metaheuristic to escape local minima
- Based on
  - Accepting non-improving solutions
  - Changing the objective or the data
  - Combining Solutions

S.A.

Today:

- More Metaheuristics!

Meta-heuristics

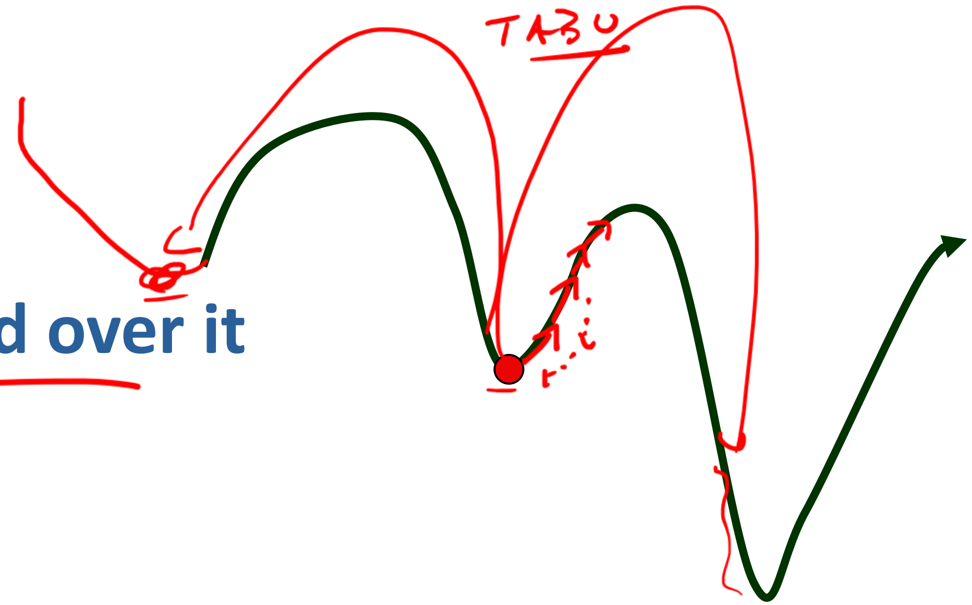


Meta-heuristics: An Incomplete Survey

- 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
    - ZCS
    - **Iterated Local Search; Restarts**
    - GRASP
  - Combine different solutions
    - Evolutionary Algorithms
    - Path Re-linking
    - Ant Colony Optimisation

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

"CONTROLLED RANDOM START"

- 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

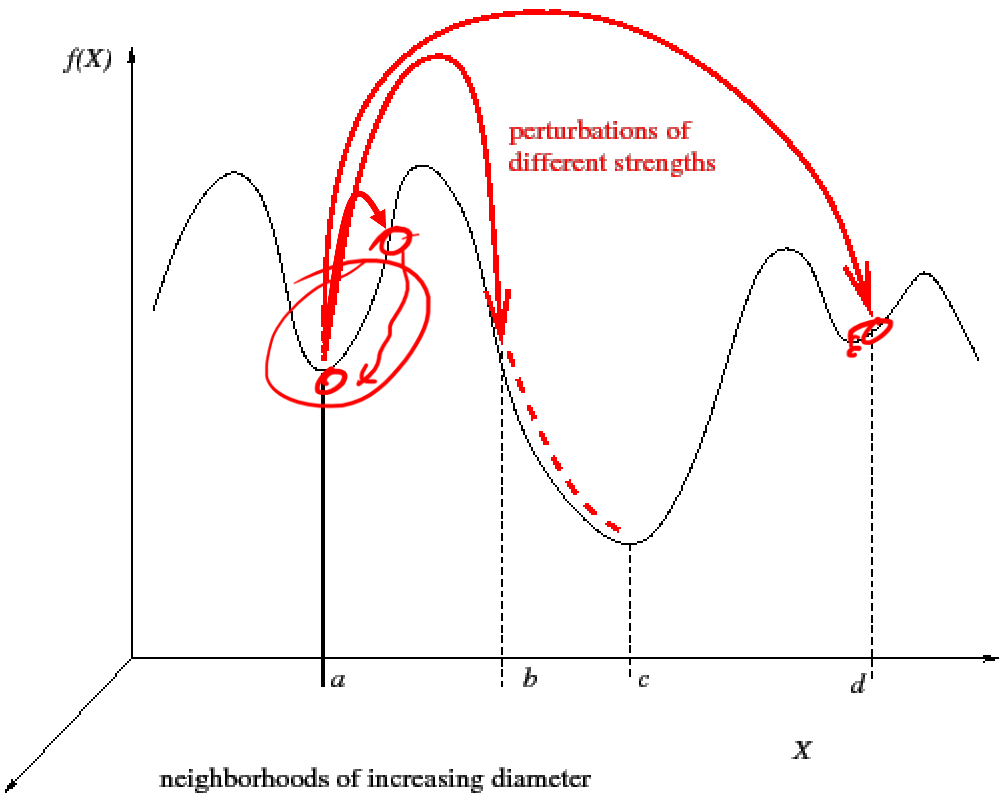


Diagram credit: Wikipedia

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)

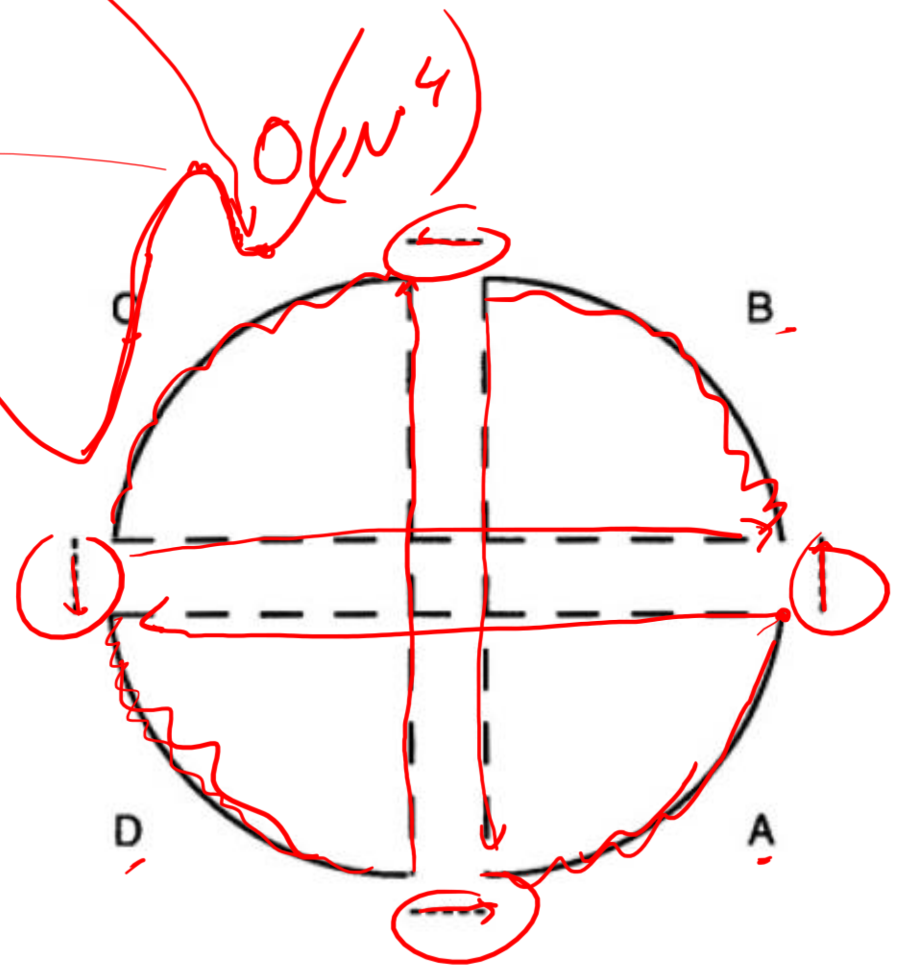
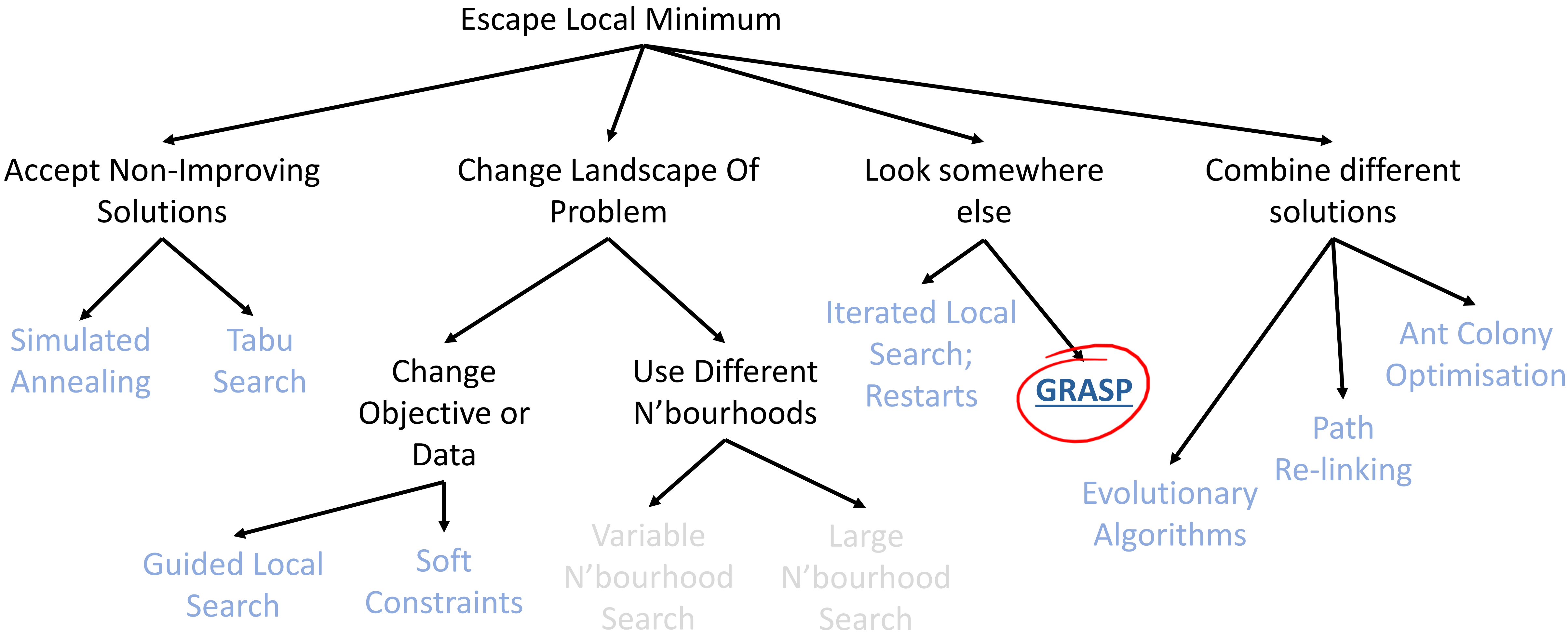


Image credit: Lourenço H.R., Martin O.C., Stützle T. (2003) Iterated Local Search

Metaheuristics



GRASP

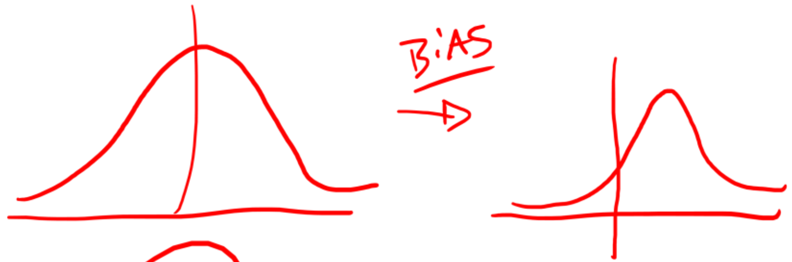
Greedy **R**andomised **A**scent **P**rocedure

- 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** URP
  - Uses a Reduced Candidate List (RCL) to guide insertion

≠ ML FEAT.

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

GRASP



Input: α

Soln = *empty* ← DIF.

MOST M.H.

repeat

GREEDY → *FeatCost*$_i$ = `CostToAddFeature` (Soln, *i*) for *i* not in Soln

minCost = **min**(*FeatCost*)

maxCost = **max**(*FeatCost*)

RCL = {} RED. CAND. LIST

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)

← UNIF SAMPLING OVER RCL

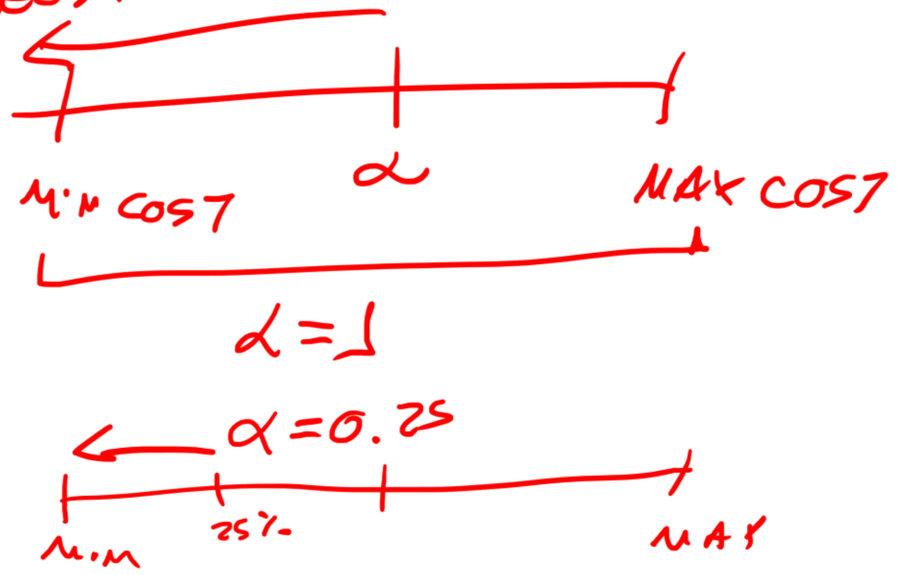
REJECTION SAMPLING

For TSP:

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

$\alpha \to 0$ GREEDY

$\alpha = 1$ RANDOM ALG



GRASP

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

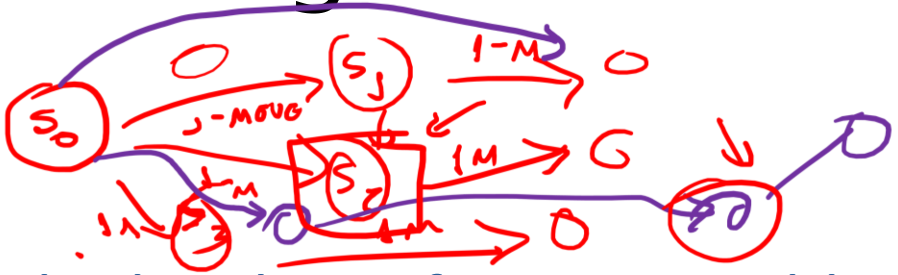
- Again, methods to adapt α to problem instance

- 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

- 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 AZ
      - **Variable N'bourhood Search** VNS ← MUCH MORE OFTEN
      - Large N'bourhood Search
  - Look somewhere else
    - ILS Iterated Local Search; Restarts
    - GRASP
  - Combine different solutions
    - Evolutionary Algorithms
    - Path Re-linking
    - Ant Colony Optimisation

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.

GLOBAL MIN
↓
SAME ON ALL NGH

LOCAL MIN
MIGHT CHANGE

Very successful method across multiple domains

Variable Neighbourhood Search

S = `initialSolution` ()

k = 0

1 — n

0 — n - 1

while k < *m*

if `LocalSearch`(S, $\mathcal{N}_k$) improved the solution S:

k = 0

else

k = k + 1

Variable Neighbourhood Search

INTERSECTION

E.g. neighbourhoods for VRP

1) 1-move (move 1 visit to another position) $O(N)$
2) 1-1 swap (swap visits in 2 routes) $O(N^2)$
3) 2-2 swap (swap 2 visits between 2 routes)
4) Or-opt size 2 (move chain of length 2 – forwards and backwards – anywhere)

4a) 1-move

5) Or-opt size 3 (chain length 3)
6) Tail exchange (swap final portion of routes)
7) 2-opt $O(N^2)$
8) 3-opt $O(N^3)$

$N_i \not\subset N_{i-1}$

$N_i \cap N_{i-1} \neq \emptyset$

NO REQ

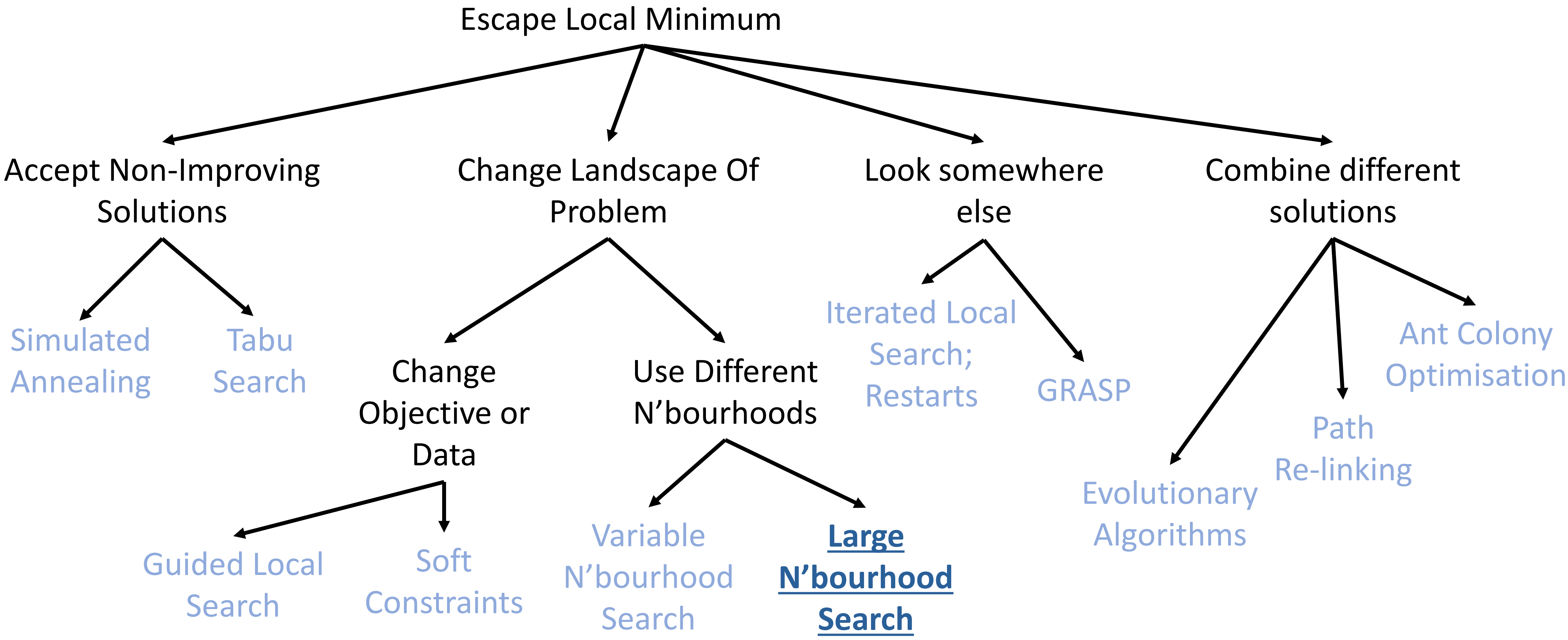
Variable Neighbourhood Search

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



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:

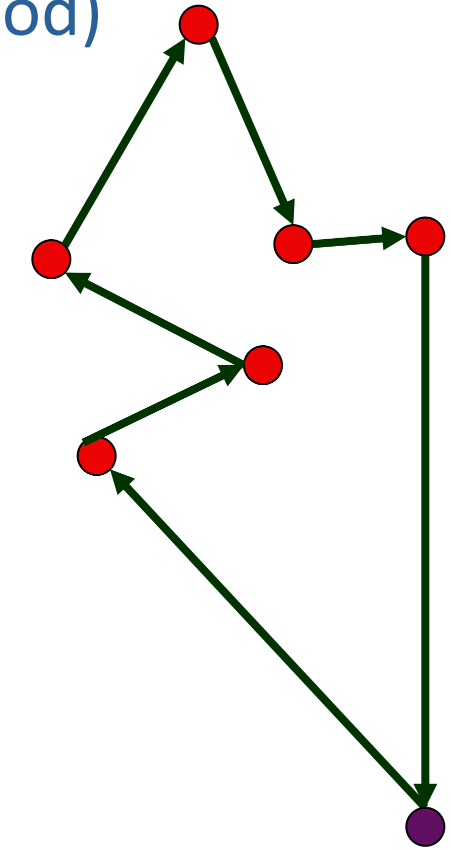
- Remove some visits
- Move them to the "unassigned" lists

Large Neighbourhood Search

LARGEST DIST = LARGEST ARC

Destroy part of the solution (Select method)

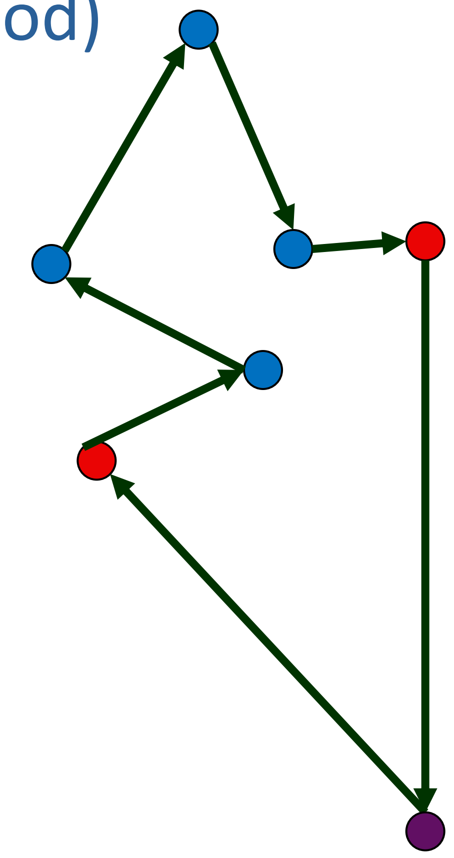
- Examples
- Remove a sequence of visits



Large Neighbourhood Search

Destroy part of the solution (Select method)

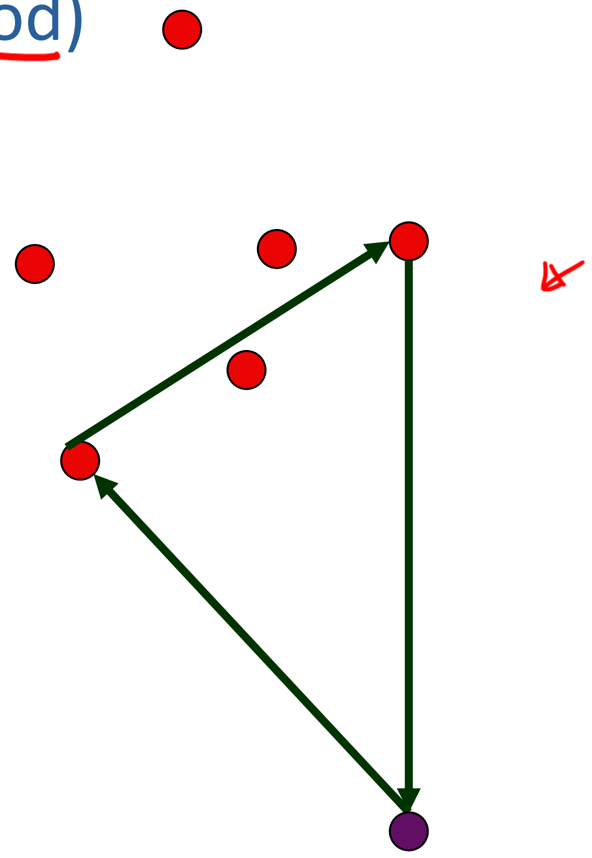
- Examples
- Remove a sequence of visits



Large Neighbourhood Search

Destroy part of the solution (Select method)

- Examples
- Remove a sequence of visits



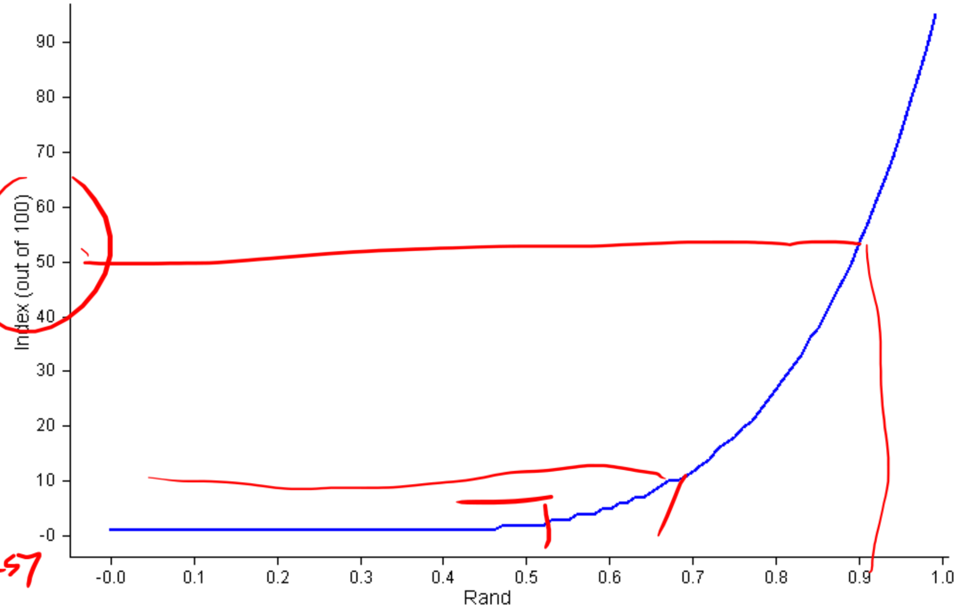
Large Neighbourhood Search

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
- $r = n * (\text{uniform}(0,1))^y$ 4-
- $\boxed{y \sim 6}$
  - unif → (unif)^y
  - 0.56 → 0.016
  - 0.96 → 0.531

TOP 66% 5 WORST

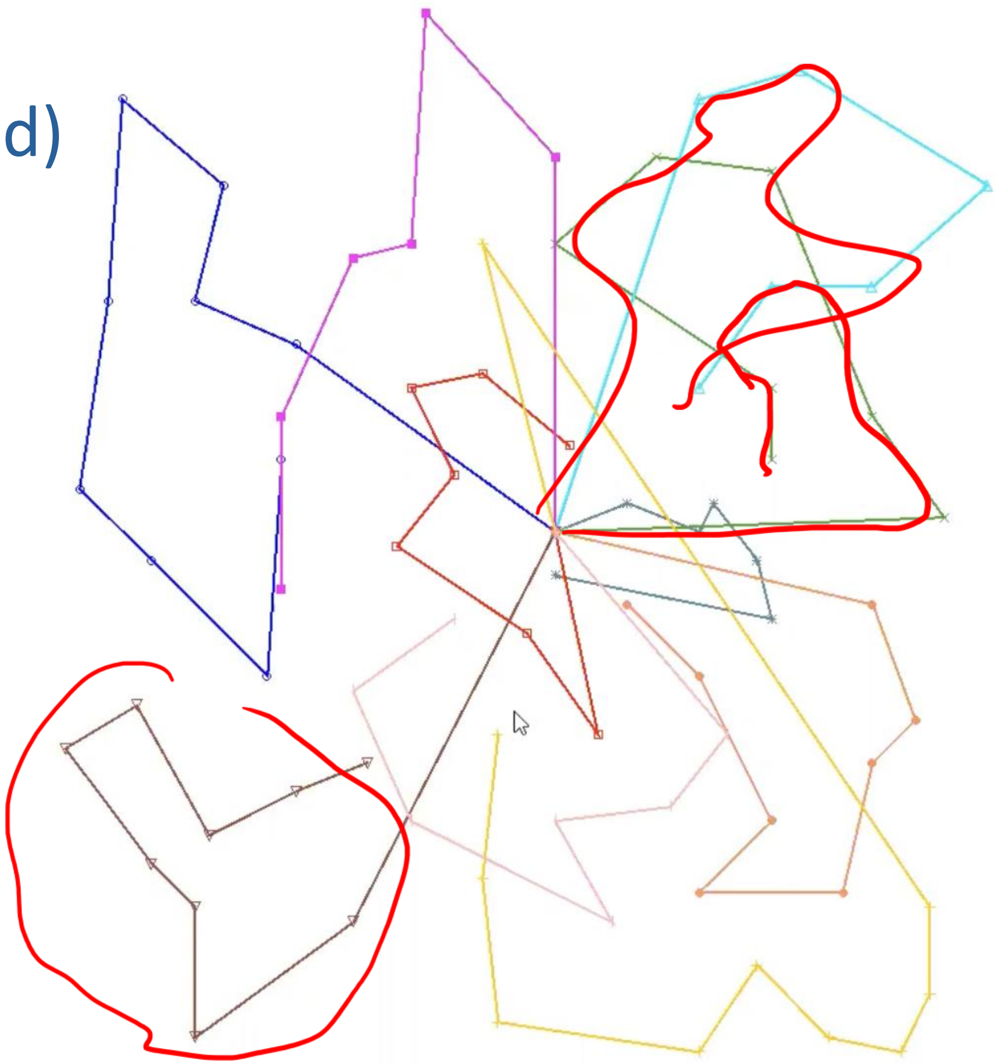


Large Neighbourhood Search

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

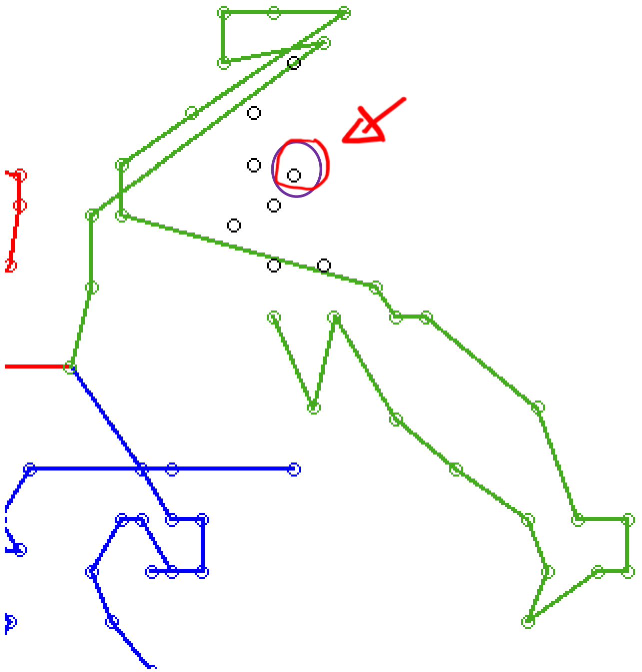
$$
\begin{aligned}
R_{ij} = \varphi C_{ij} &+ \quad \longleftarrow \text{Load} \\
\chi(|a_i - a_j|) &+ \quad \longleftarrow \text{Distance} \\
\psi(|q_i - q_j|) & \quad\quad \longleftarrow \text{Time}
\end{aligned}
$$

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)

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)

- Parameter: Max to dump
  - As a % of n?
  - As a fixed number e.g. 100 for large problems

50

- Actual number is uniform rand (5, max)

$d=$

CAN YOU FIND THE OPT d?

Large Neighbourhood Search

WE DON'T NEED OPT

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

LP
MIP

WHY USE THIS?

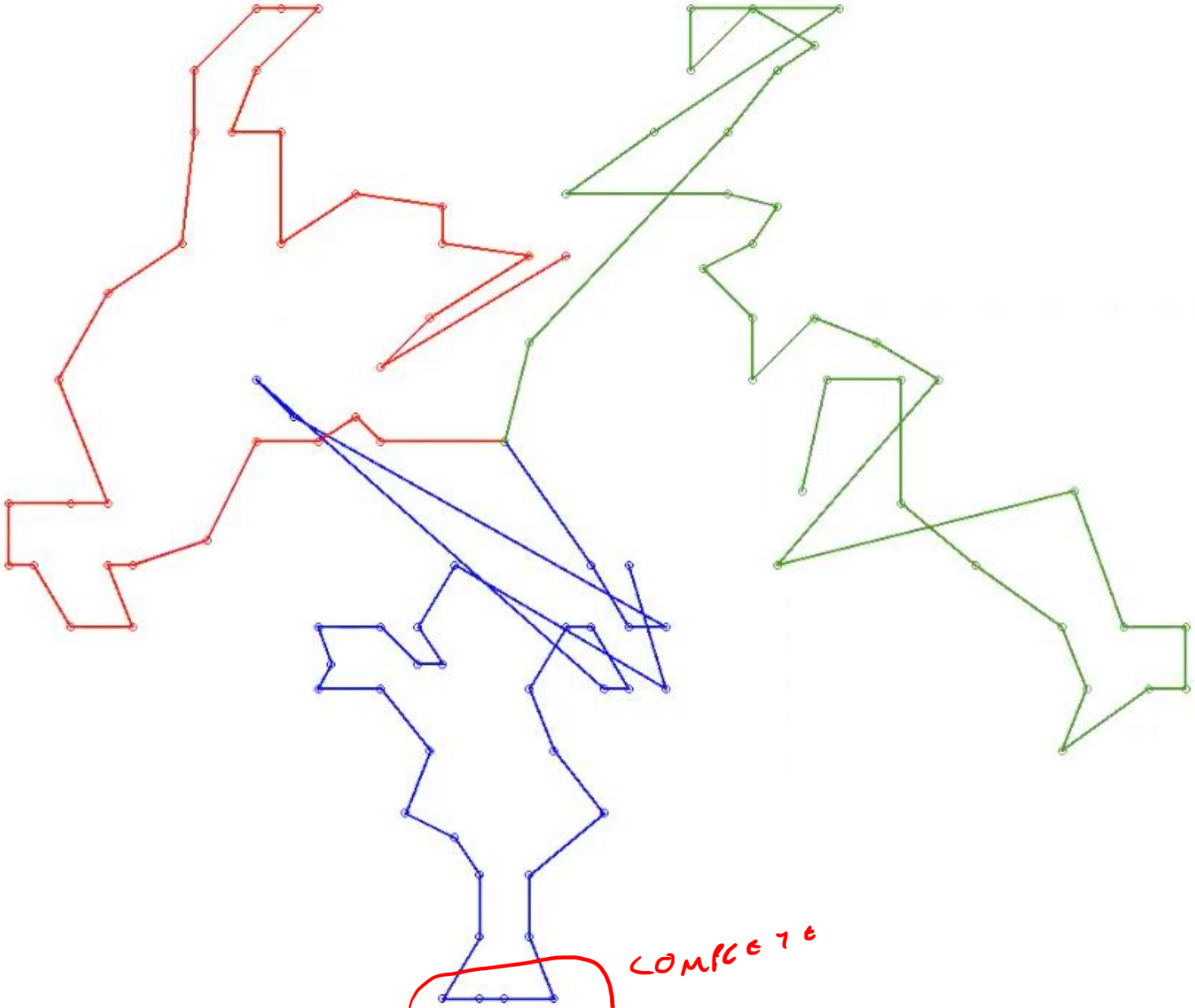
CONSTR

SIZE OF DESTRUCTION

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

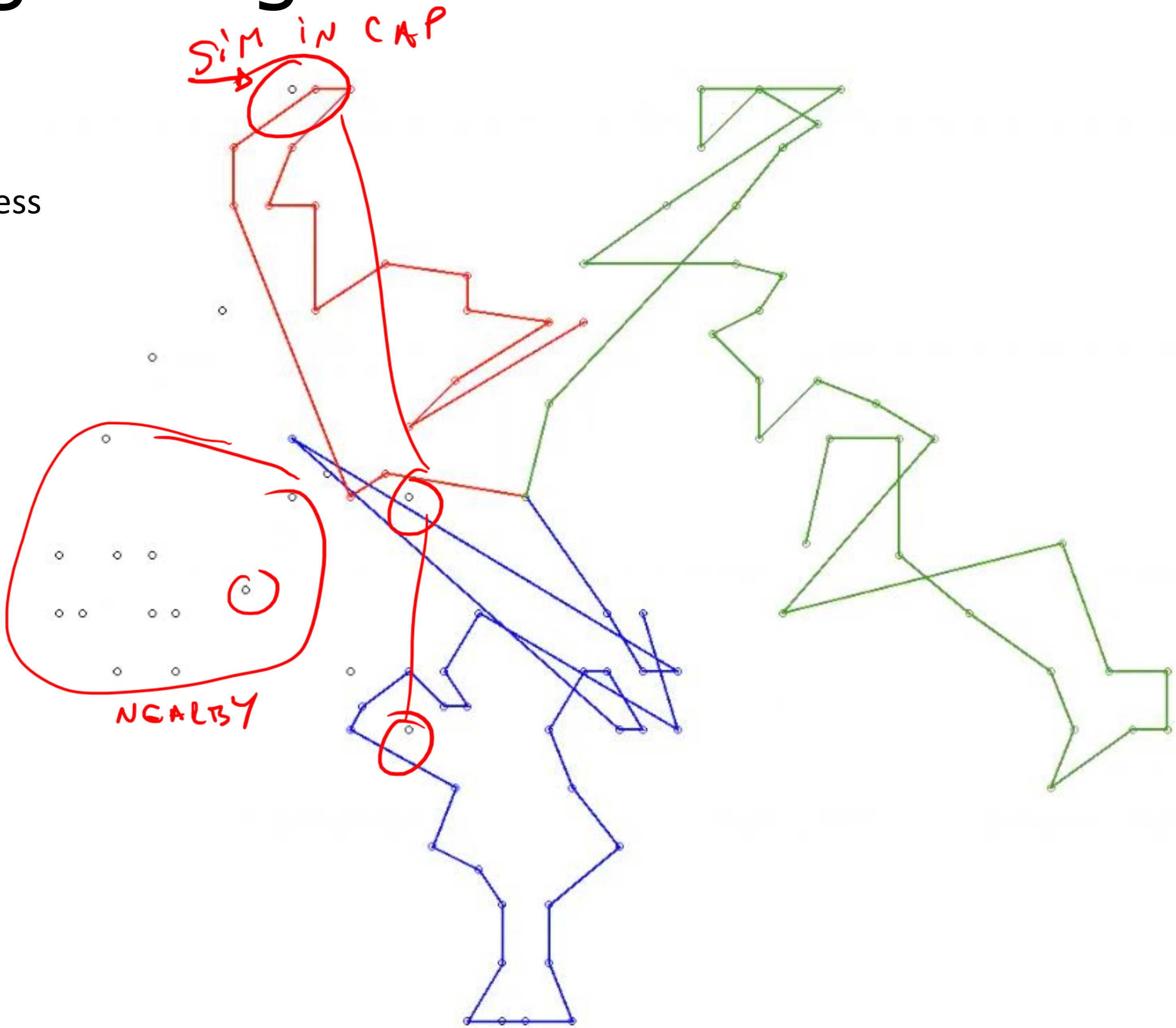
Initial solution



COMPLETE

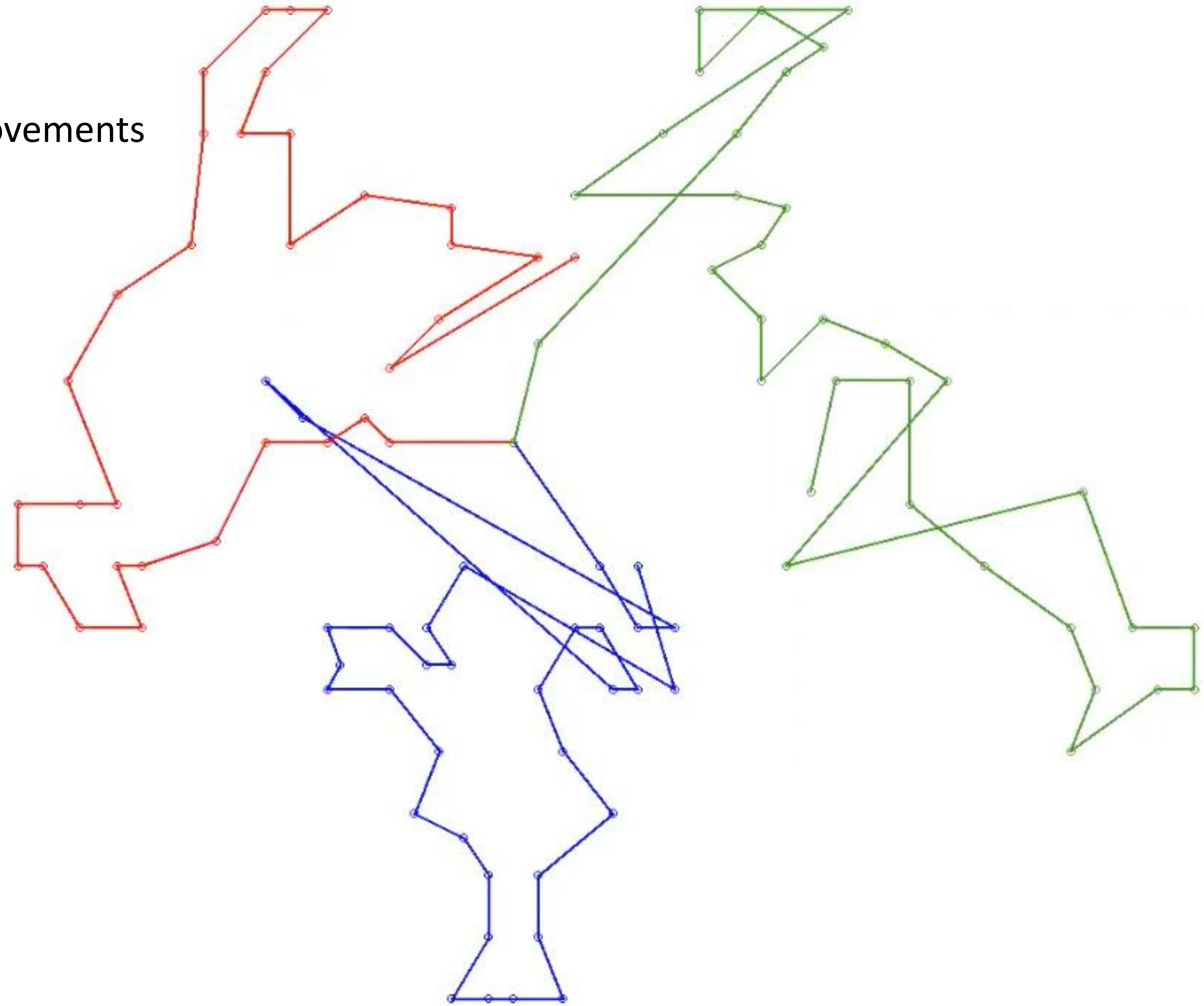
Large Neighbourhood Search – VRP

Destroy using relatedness



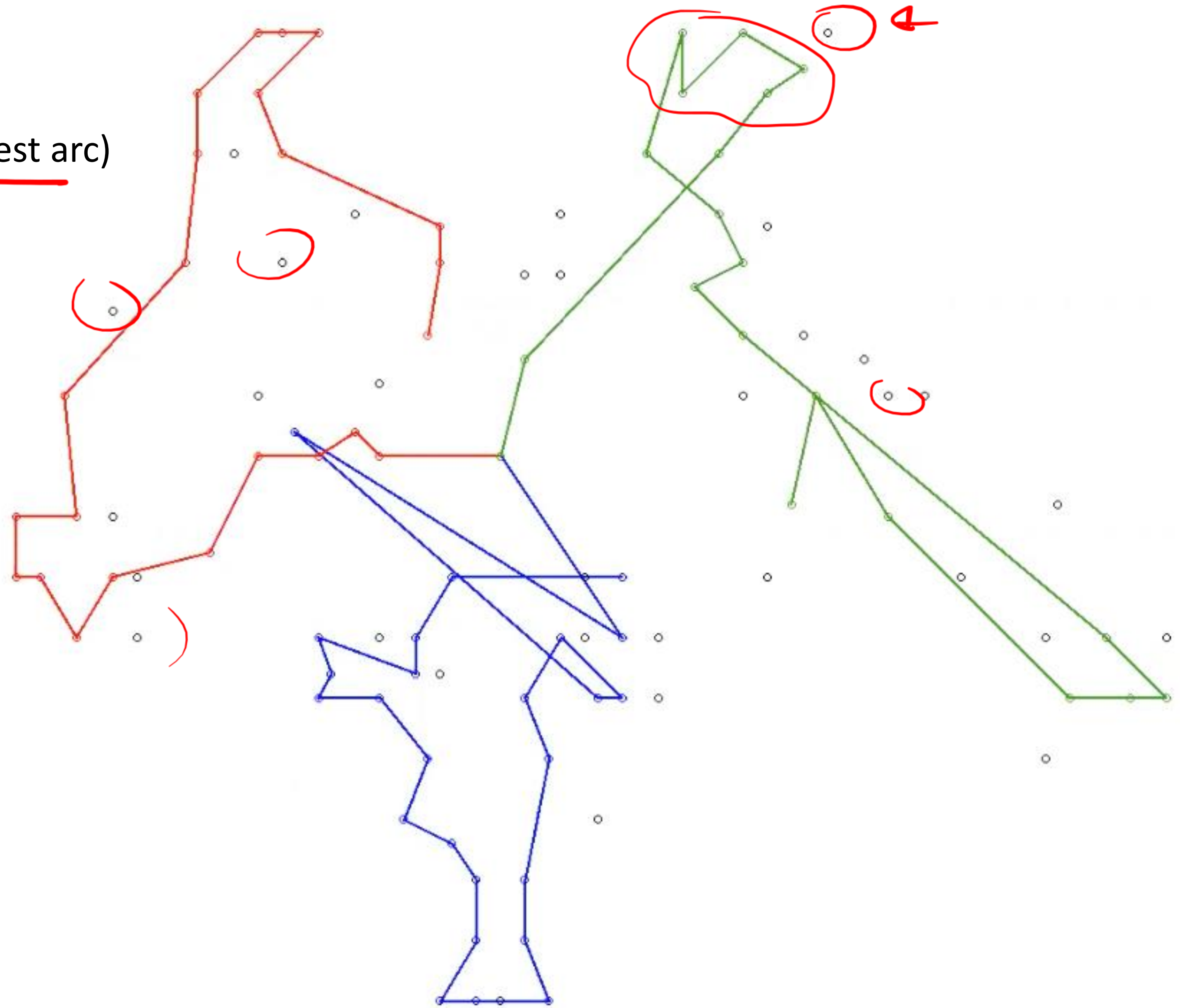
Large Neighbourhood Search – VRP

Reconstruct and no improvements



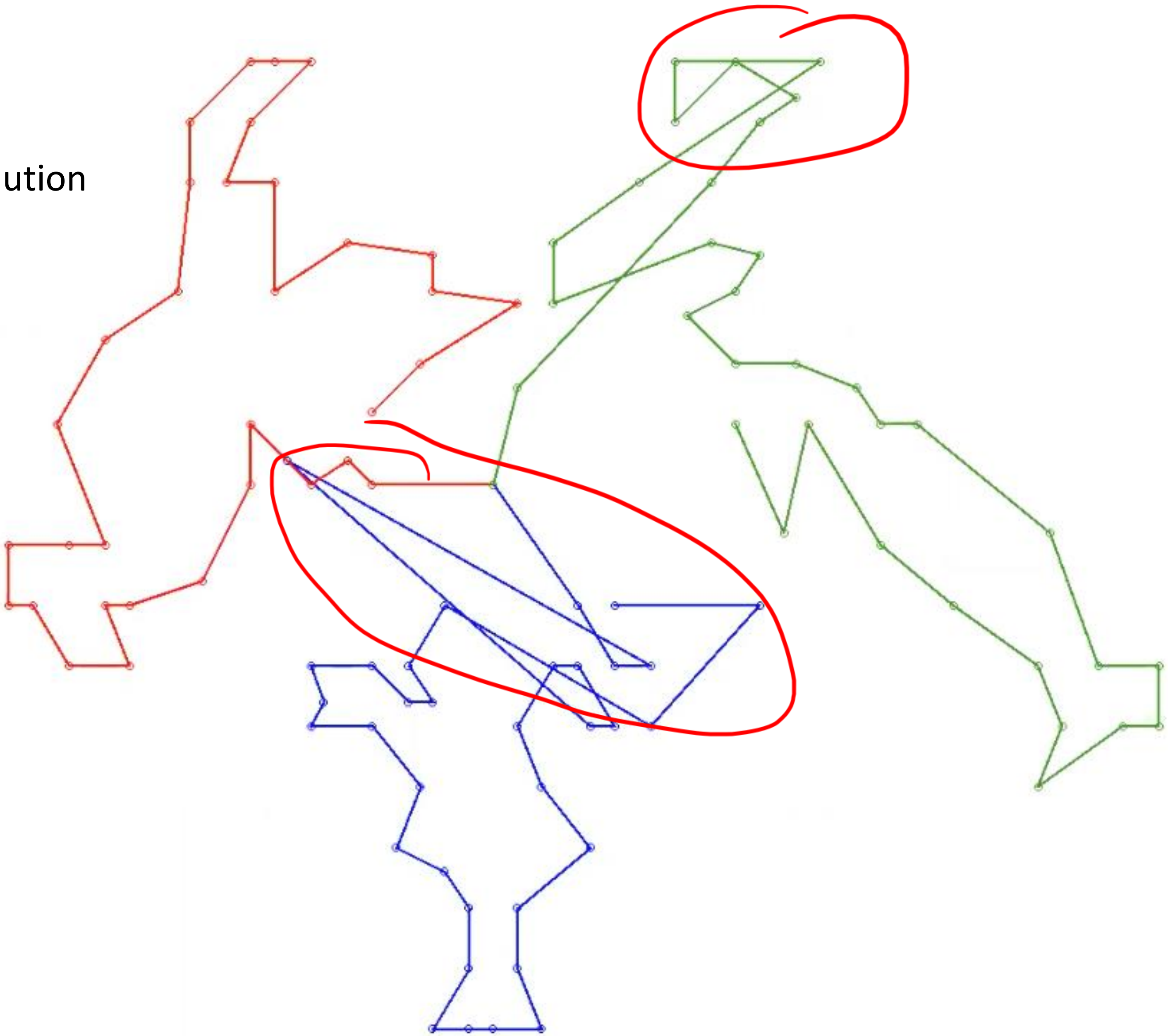
Large Neighbourhood Search – VRP

Destroy using worst (longest arc)



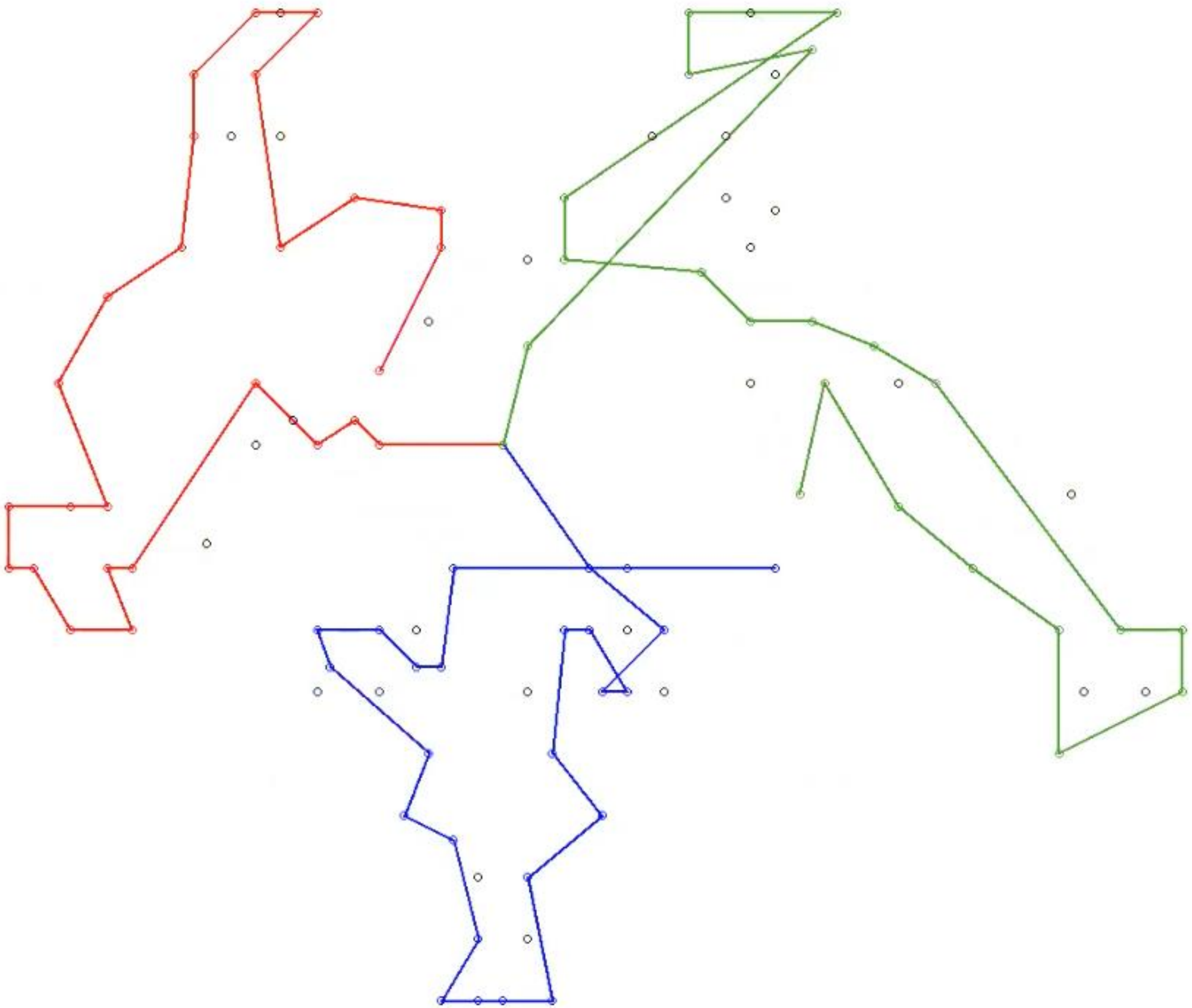
Large Neighbourhood Search – VRP

Reconstruct and better solution found



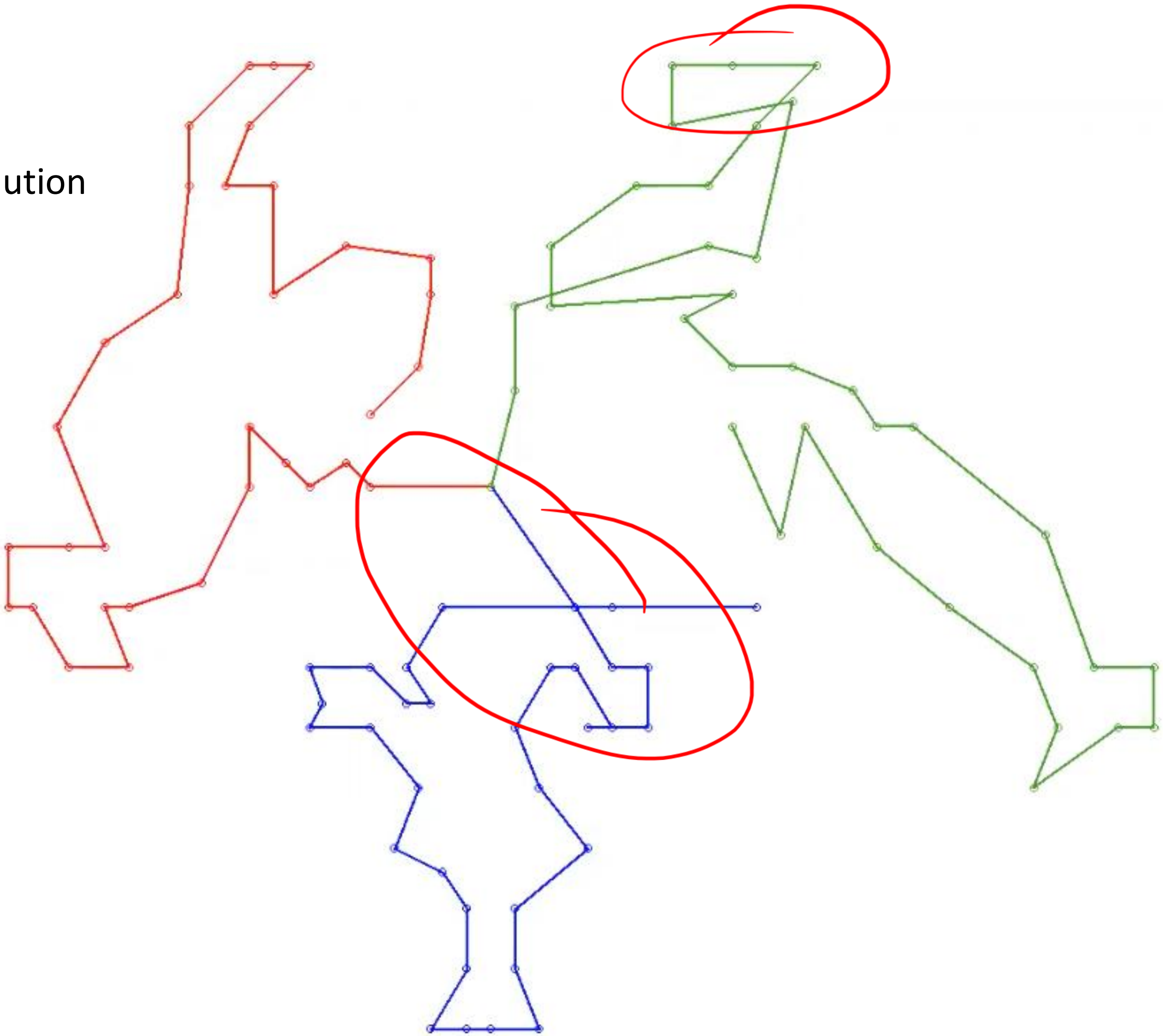
Large Neighbourhood Search – VRP

Destroy using random



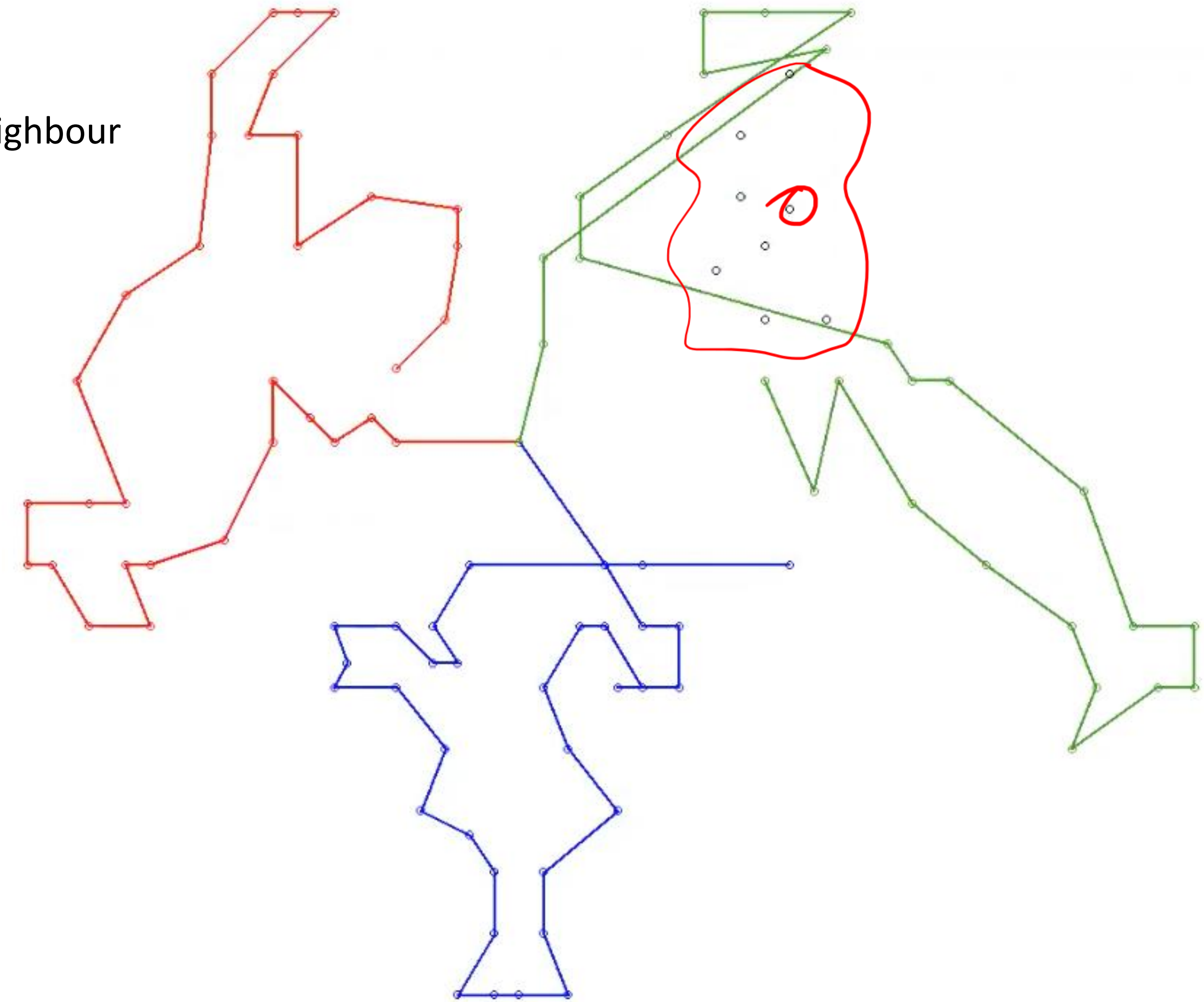
Large Neighbourhood Search – VRP

Reconstruct and better solution found



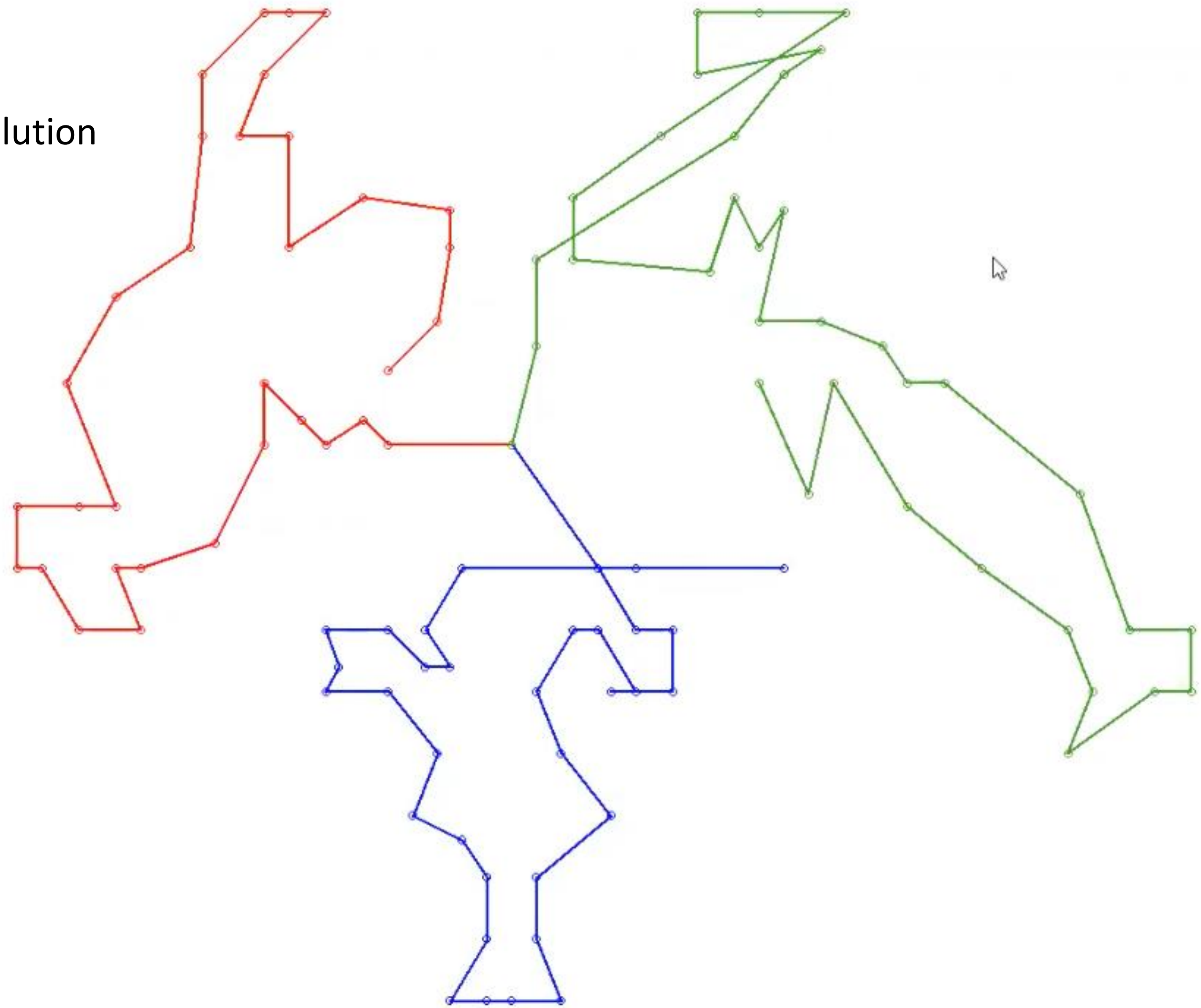
Large Neighbourhood Search – VRP

Destroy using nearest neighbour



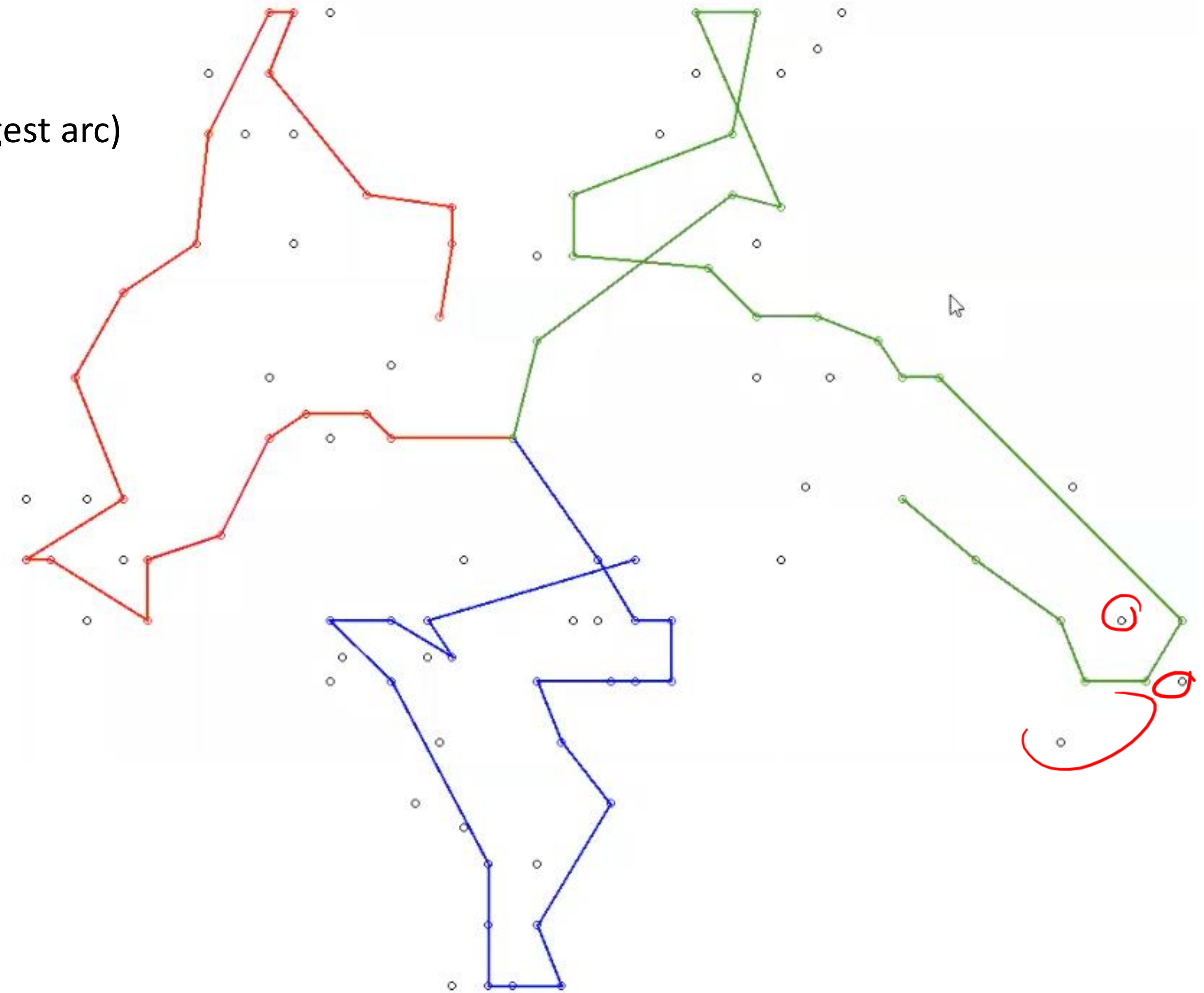
Large Neighbourhood Search – VRP

Reconstruct and better solution found



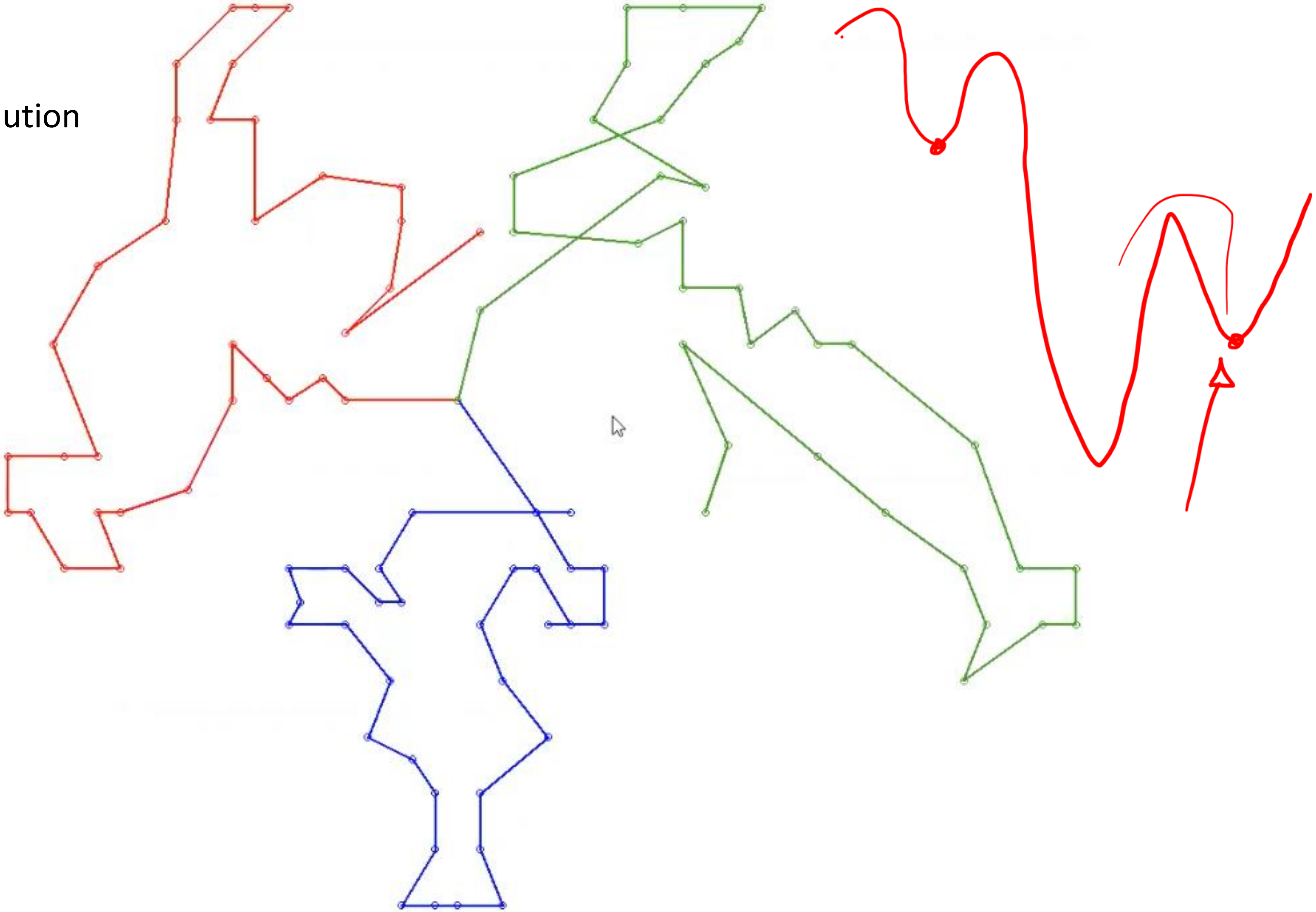
Large Neighbourhood Search – VRP

Destroy using worst (longest arc)



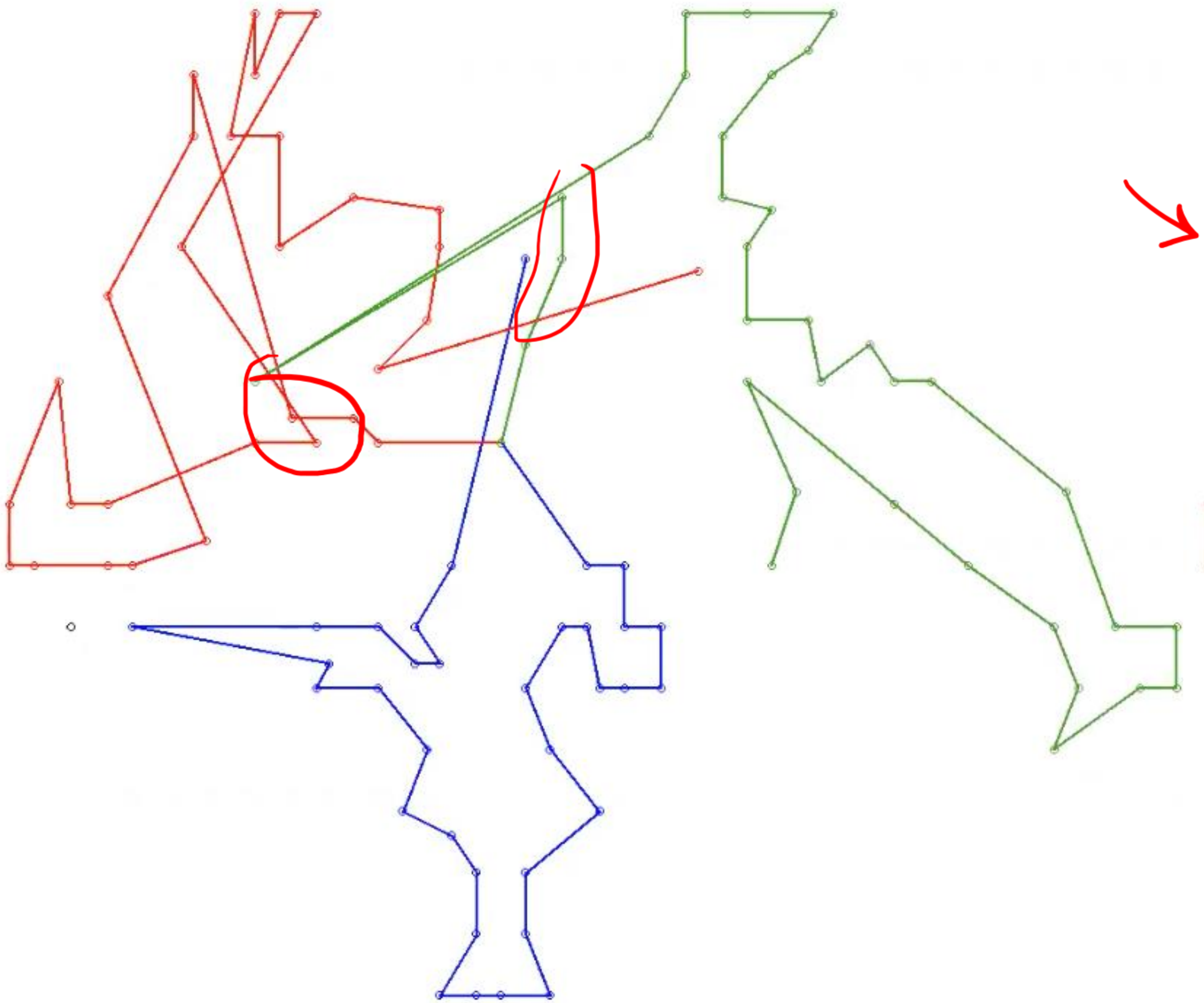
Large Neighbourhood Search – VRP

Reconstruct and better solution found

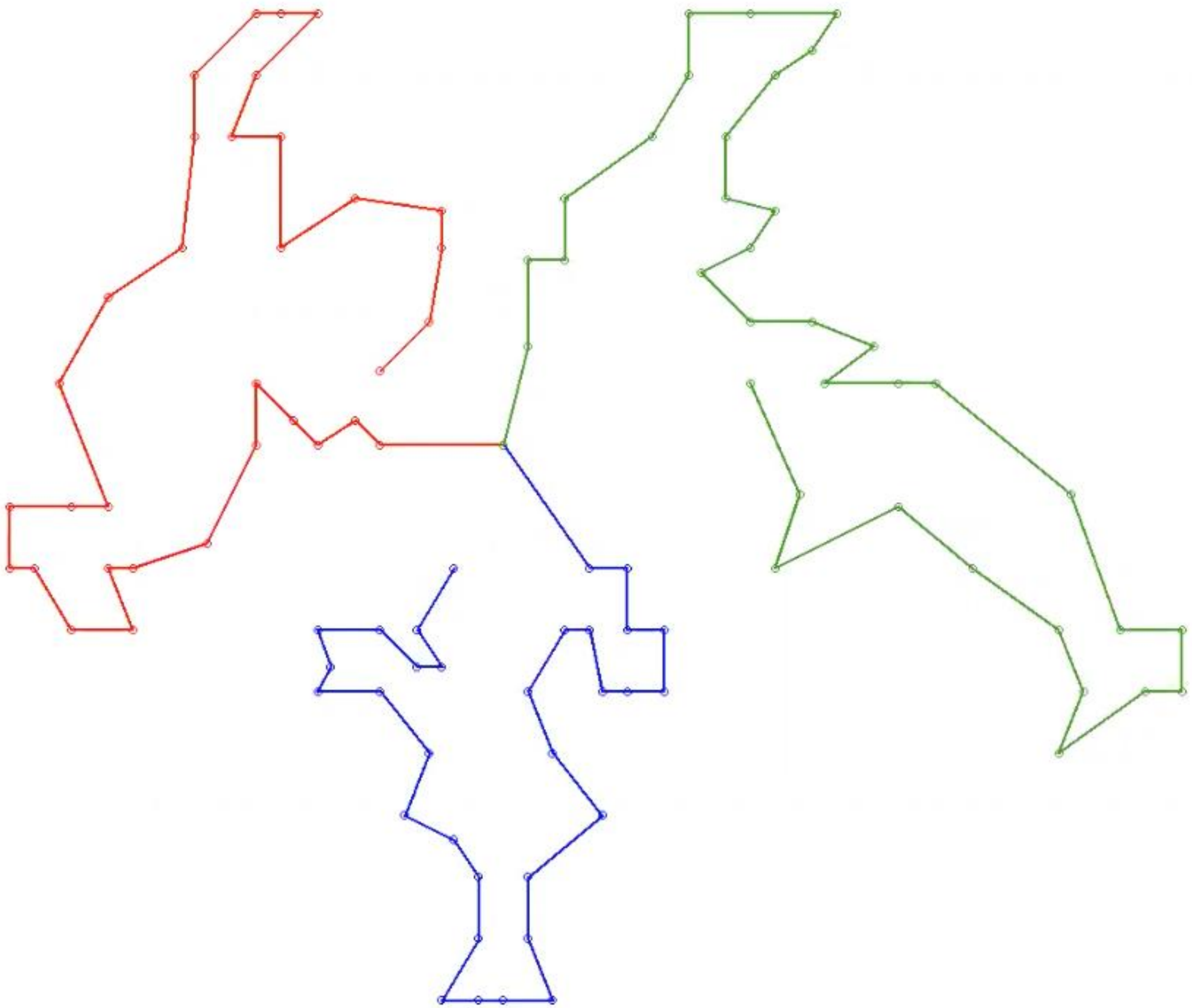


Large Neighbourhood Search – VRP

We can observe some really bad solutions during the iterations

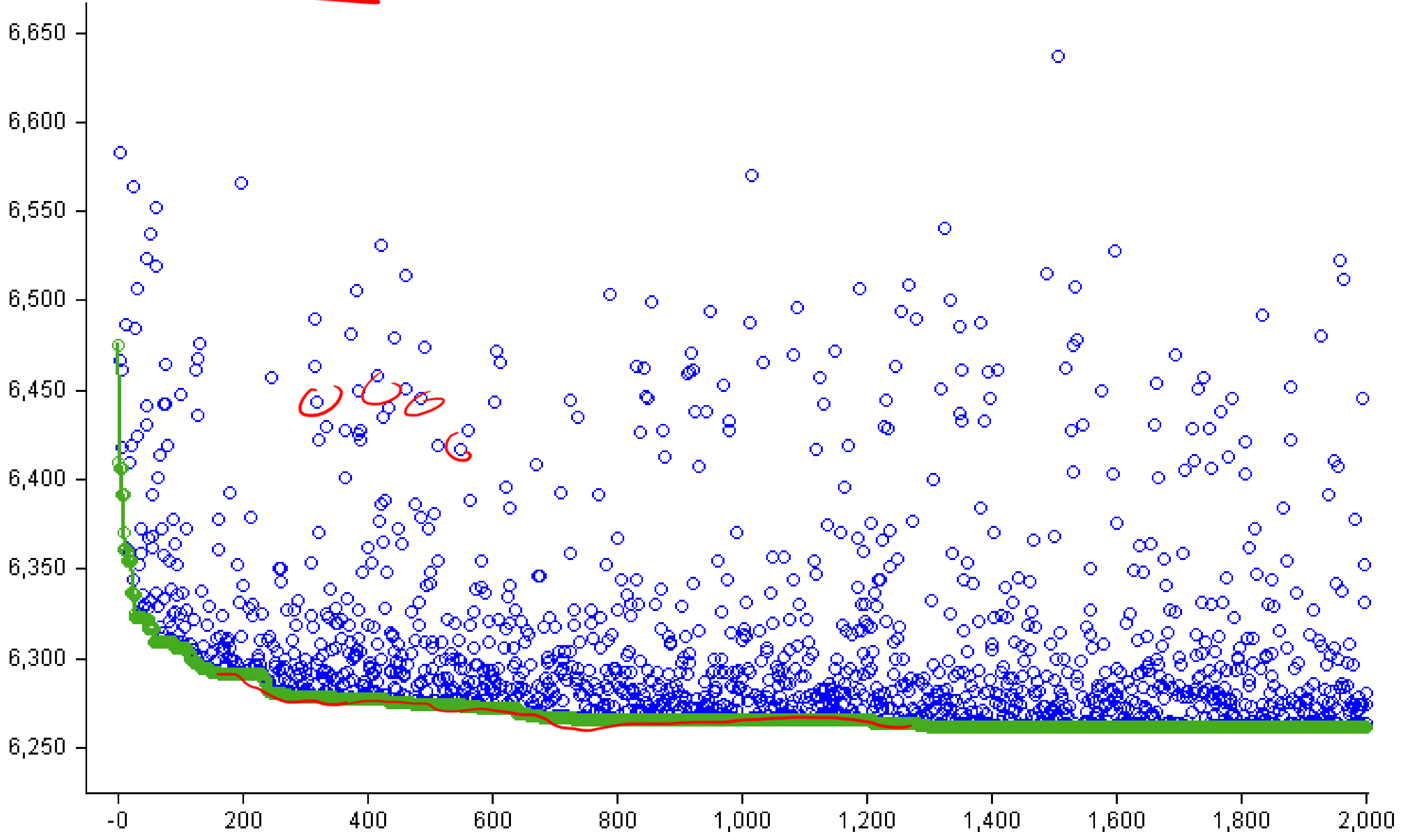


After less than 50 iterations we get this route that is the best known for this problem



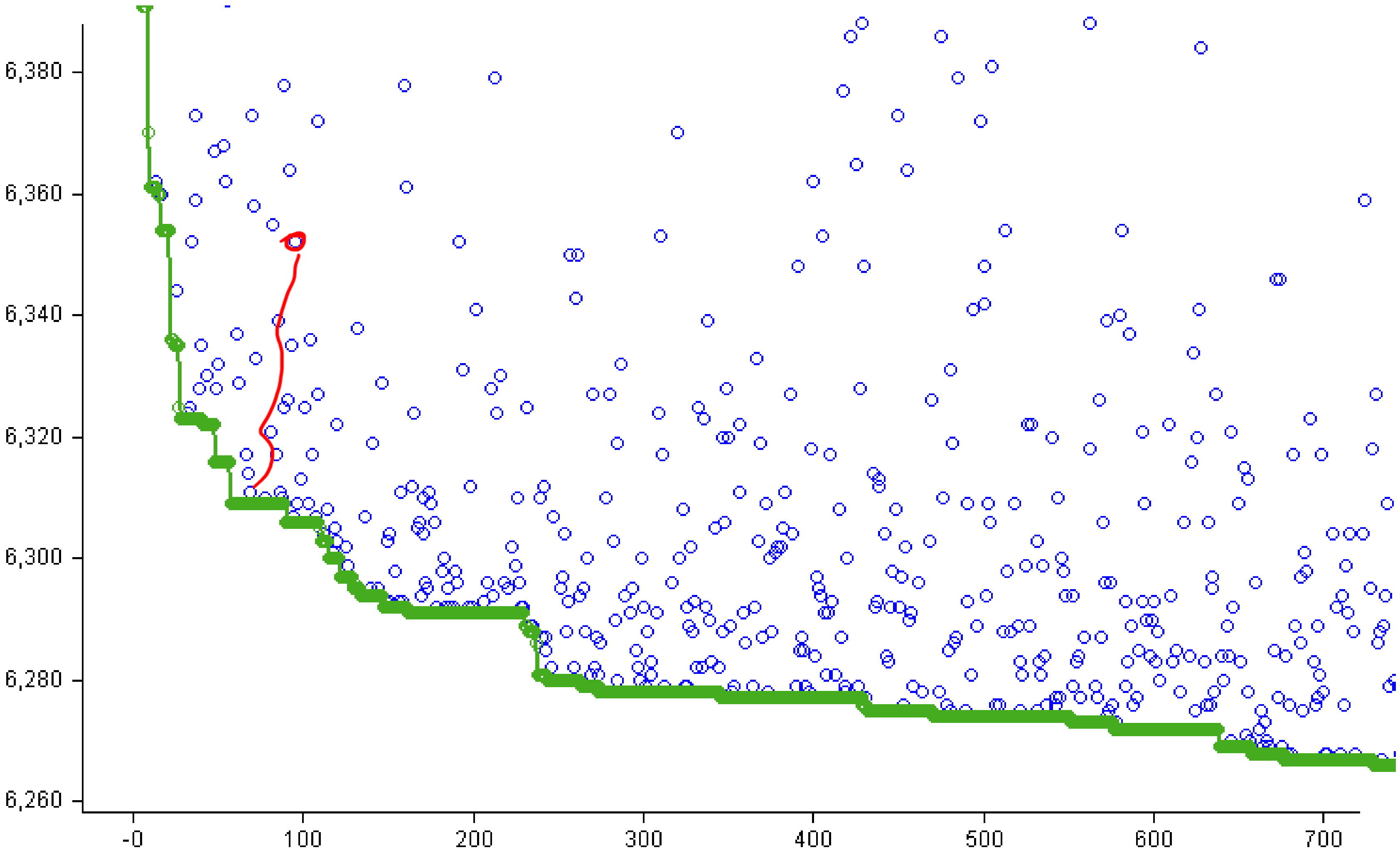
Large Neighbourhood Search

- 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

WHEN A METHOD FINDS A BETTER SOL → GIVE A REWARD

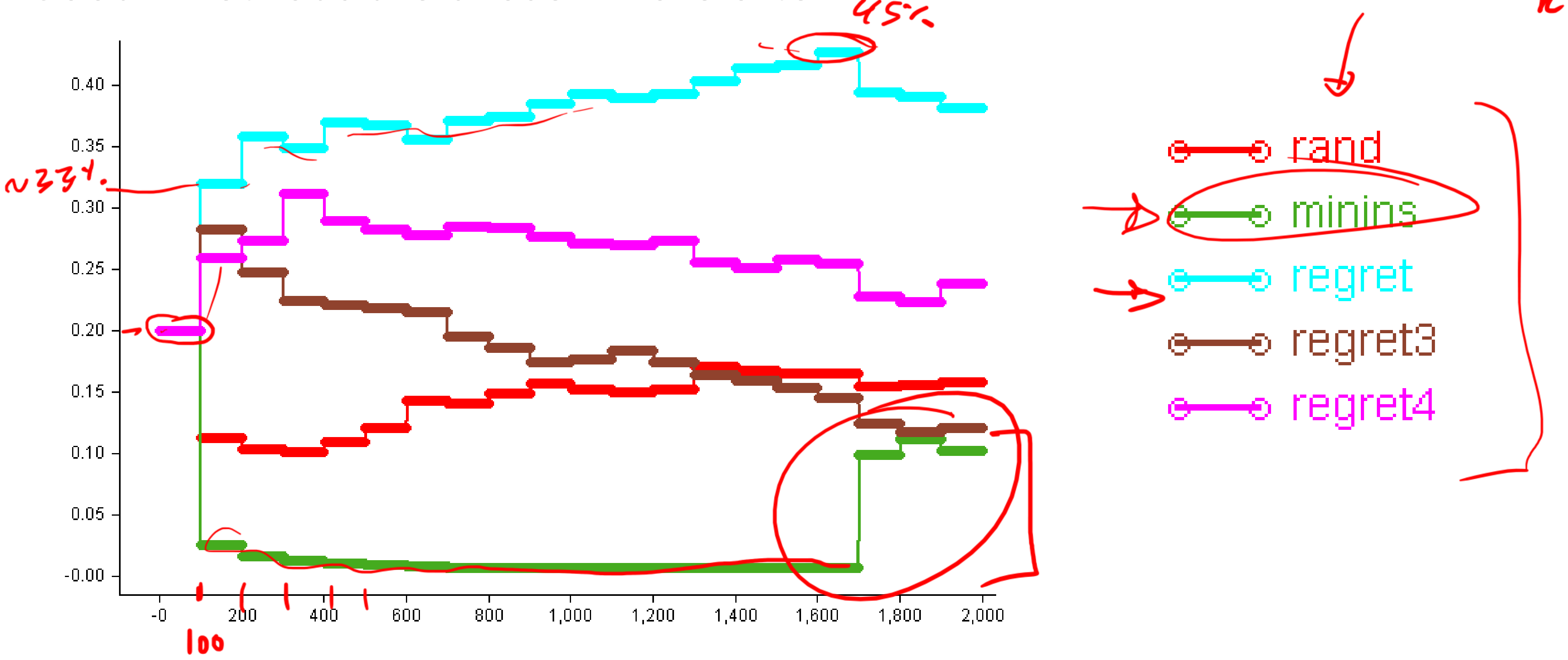
TINY REW

Adaptive Insert Method ← RECONSTR.

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

START W/ UNIF RAND.

Probability of choosing method

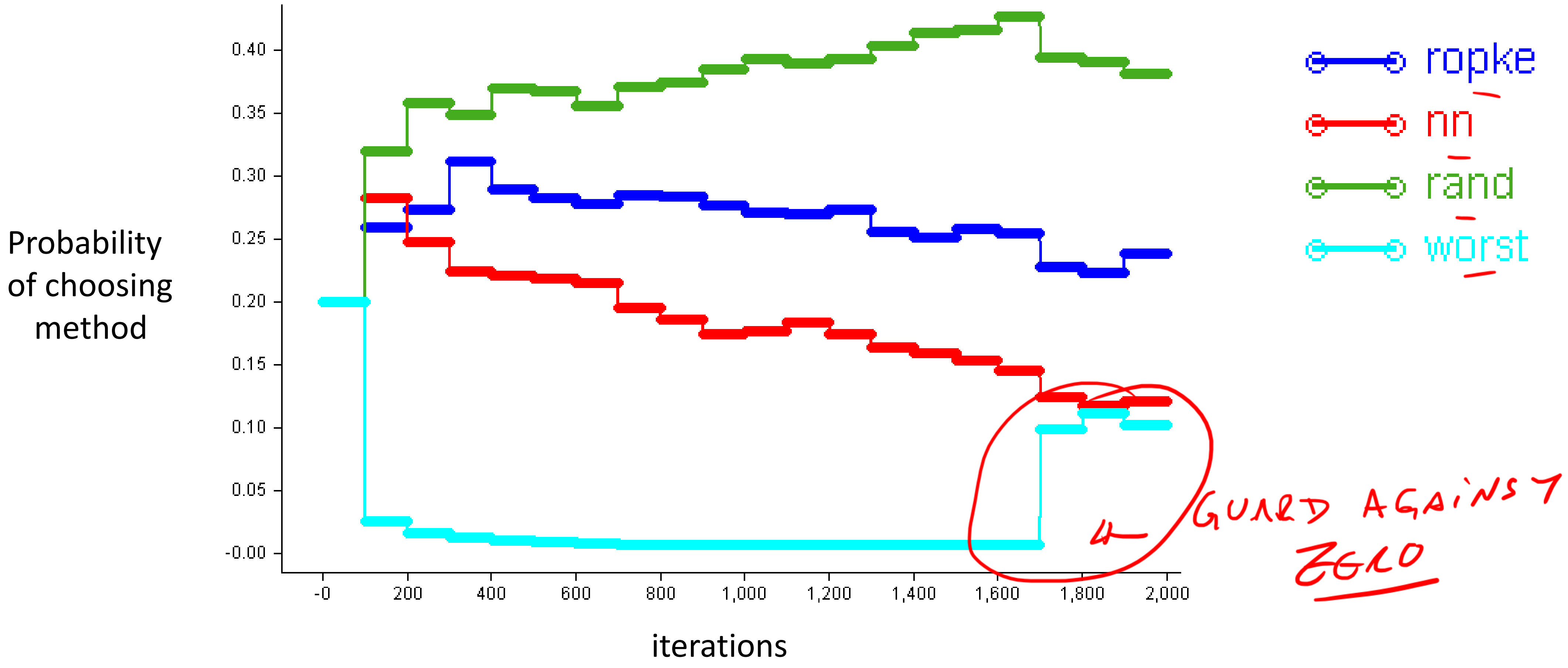


iterations

Large Neighbourhood Search

→ SIZE

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

- 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

MEMETIC
GA + L.S.

Course Outline

- Linear Programming
- Mixed-Integer Linear Programming
- Decomposition
- Convex Optimisation
- Local Search & Metaheuristics 4 2 W.K.
- Advanced Topics 4

M. PROG

Course Outline – Weeks 10 to 12

W10
- Tue Oct 8th: **no lecture**
- Wed 9th – Thu 10th : Meta-Heuristics Tutorial
- Fri Oct 11th : Multi-Objective Optimisation + Quiz on Meta-Heuristics

W11
- Tue Oct 15th : Stochatic Optimisation
- Wed 16th – Thu 17th : Meta-Heuristics drop-ins

Felipe's Guest Lec.

- Fri Oct 18th : Network Flows and AI Planning

W12
- Tue Oct 22nd : Review lecture
- Friday Oct 25th : Charles Gretton on Satisfiability + Quiz
- Sunday Oct 27th : Assignment 2 due