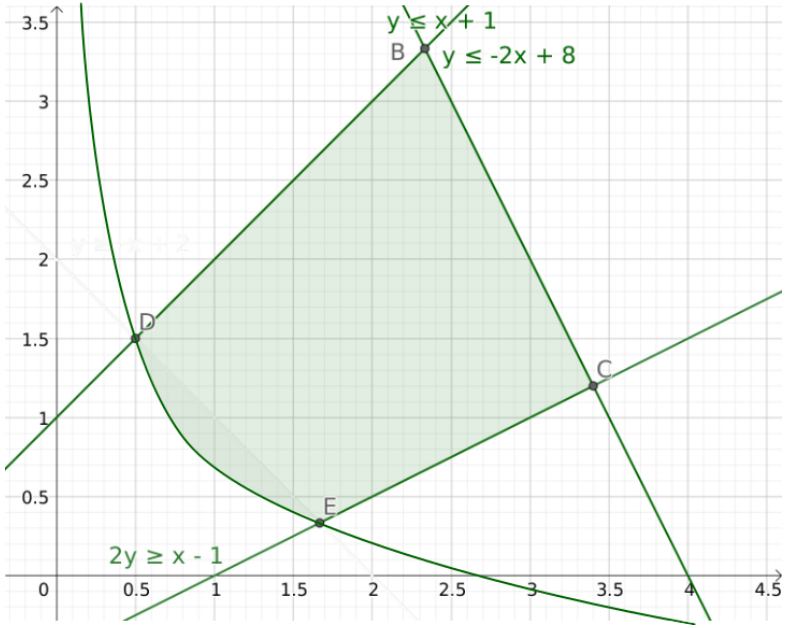
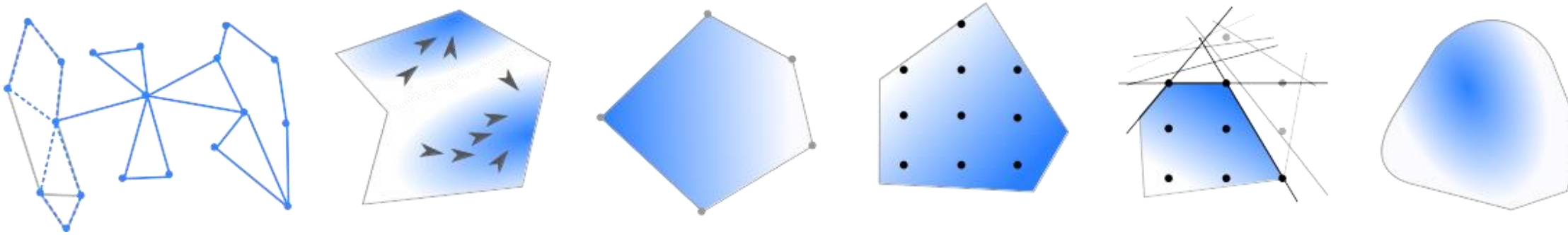
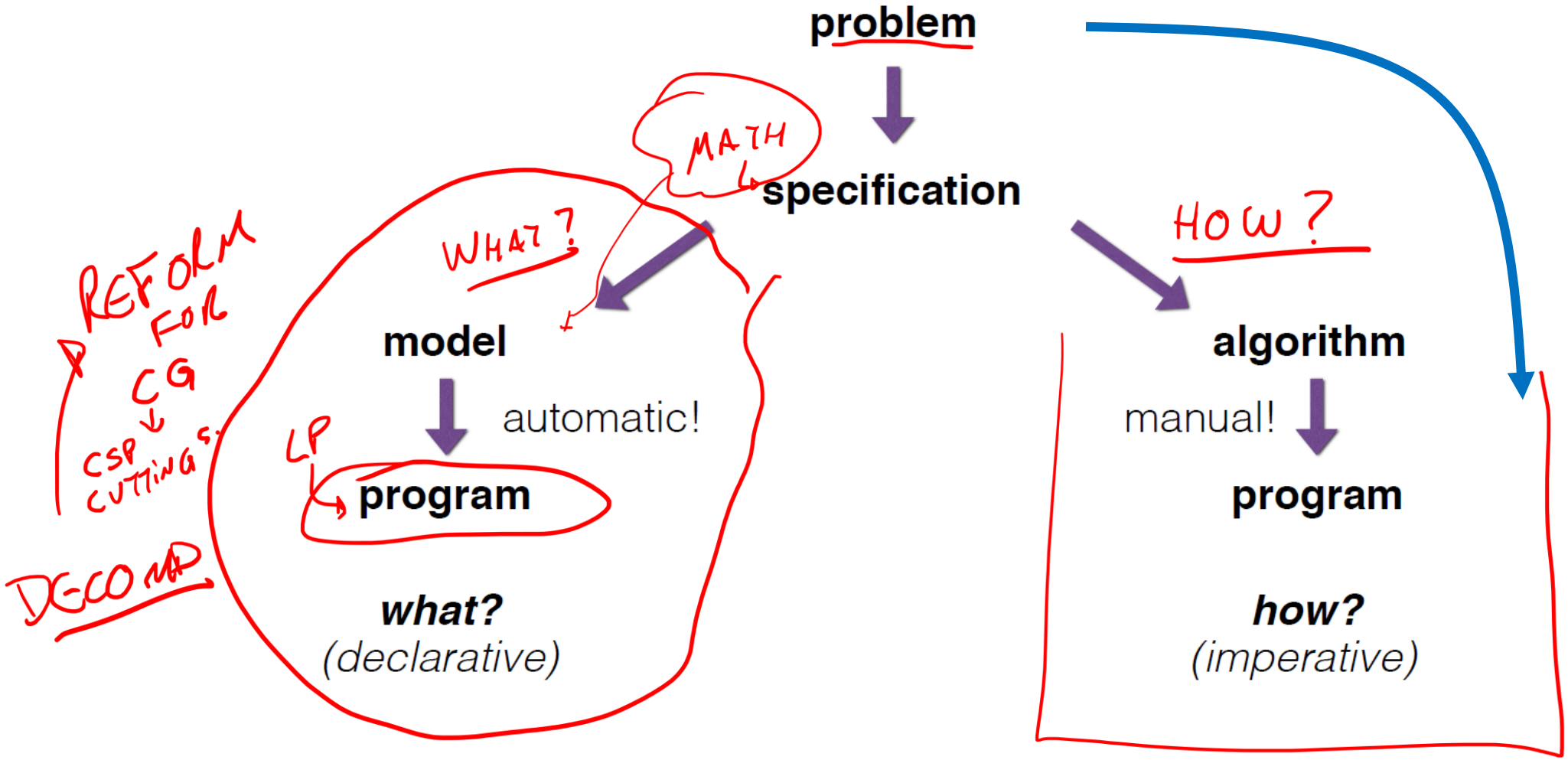
Construction Algorithms

COMP4691 / 8691





Previously on COMP4691(8691)

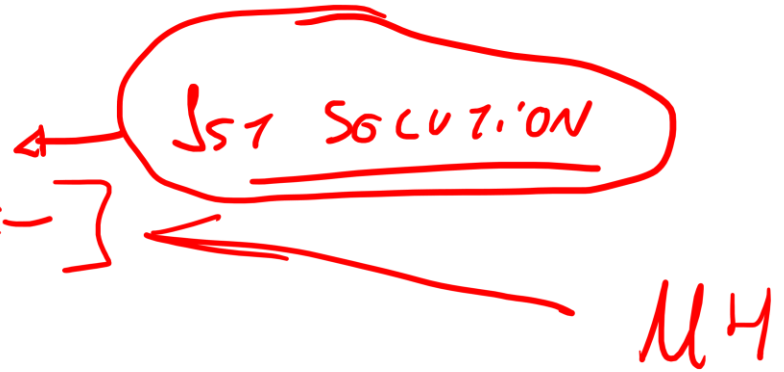


Previously on COMP4691(8691)

- Looked at a model-based approach to solving (MIPs)

Now
- Look at heuristic approaches

A classic
- Construct
- Improve



Today:
- Construct

SCHEDUCING
→ LNS
LARGE NEIGH SEARCH

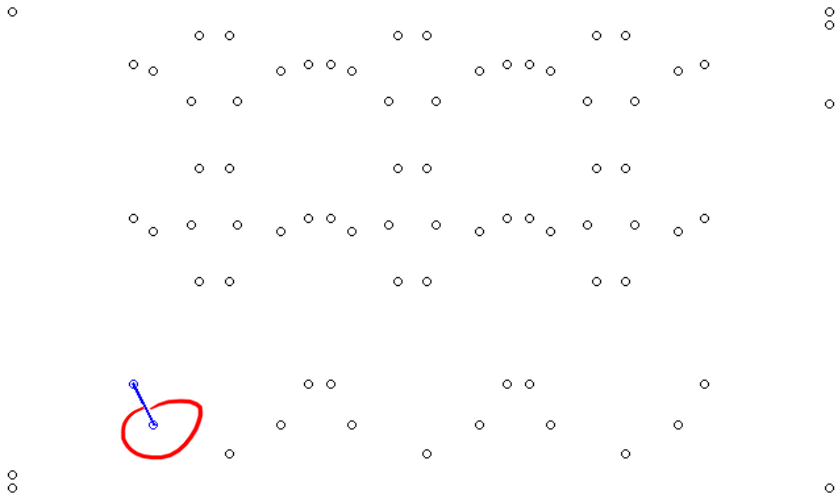
Construction algorithms

We need to start somewhere

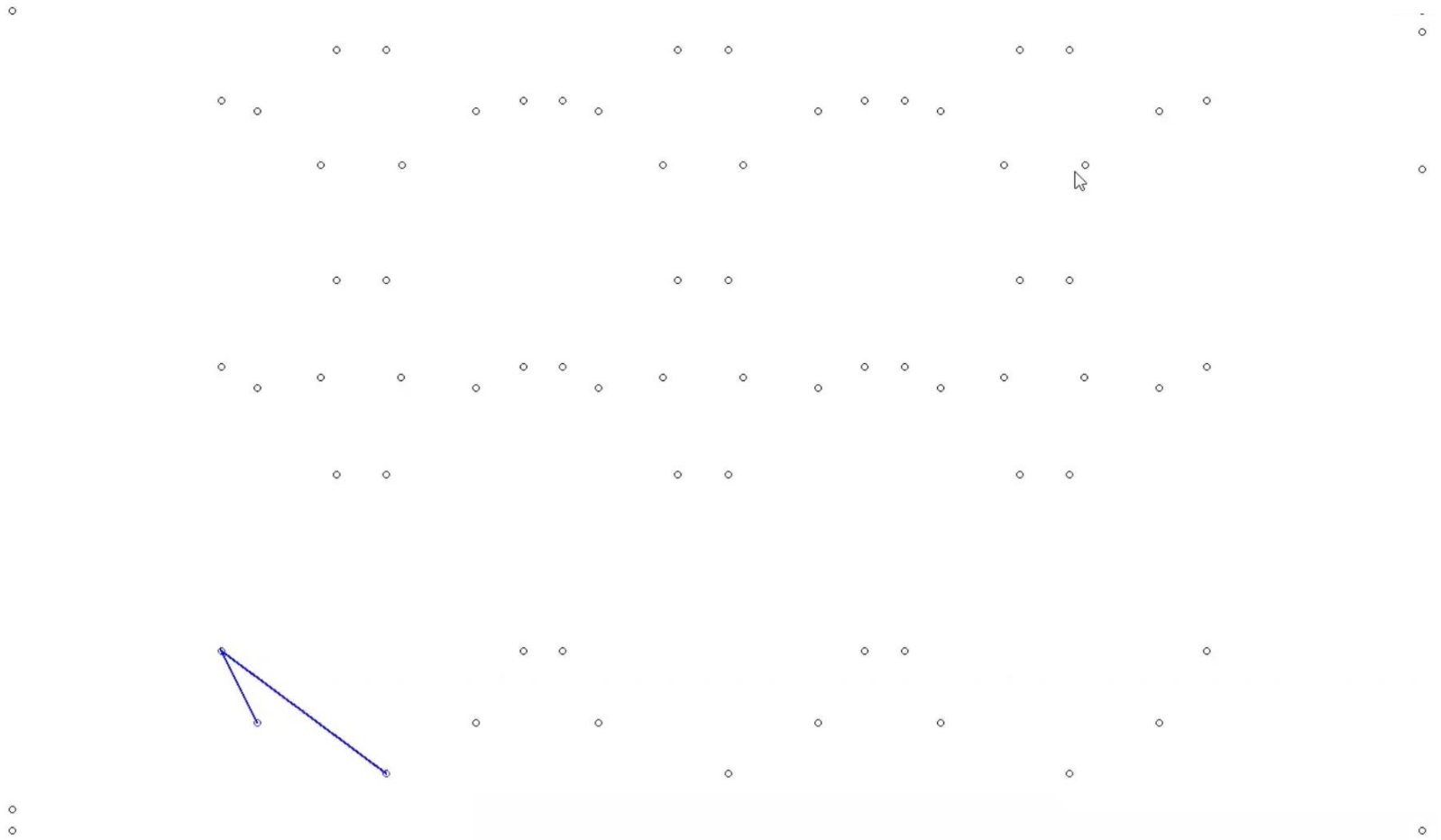
- When we first come to an instance we are looking at an empty page
  - In TSP, no cities are visited
  - In Knapsack, all the items are lying on the ground
- Initial solution often created by “constructing” one piece / element at a time

Construction algorithms – Nearest Neighbour

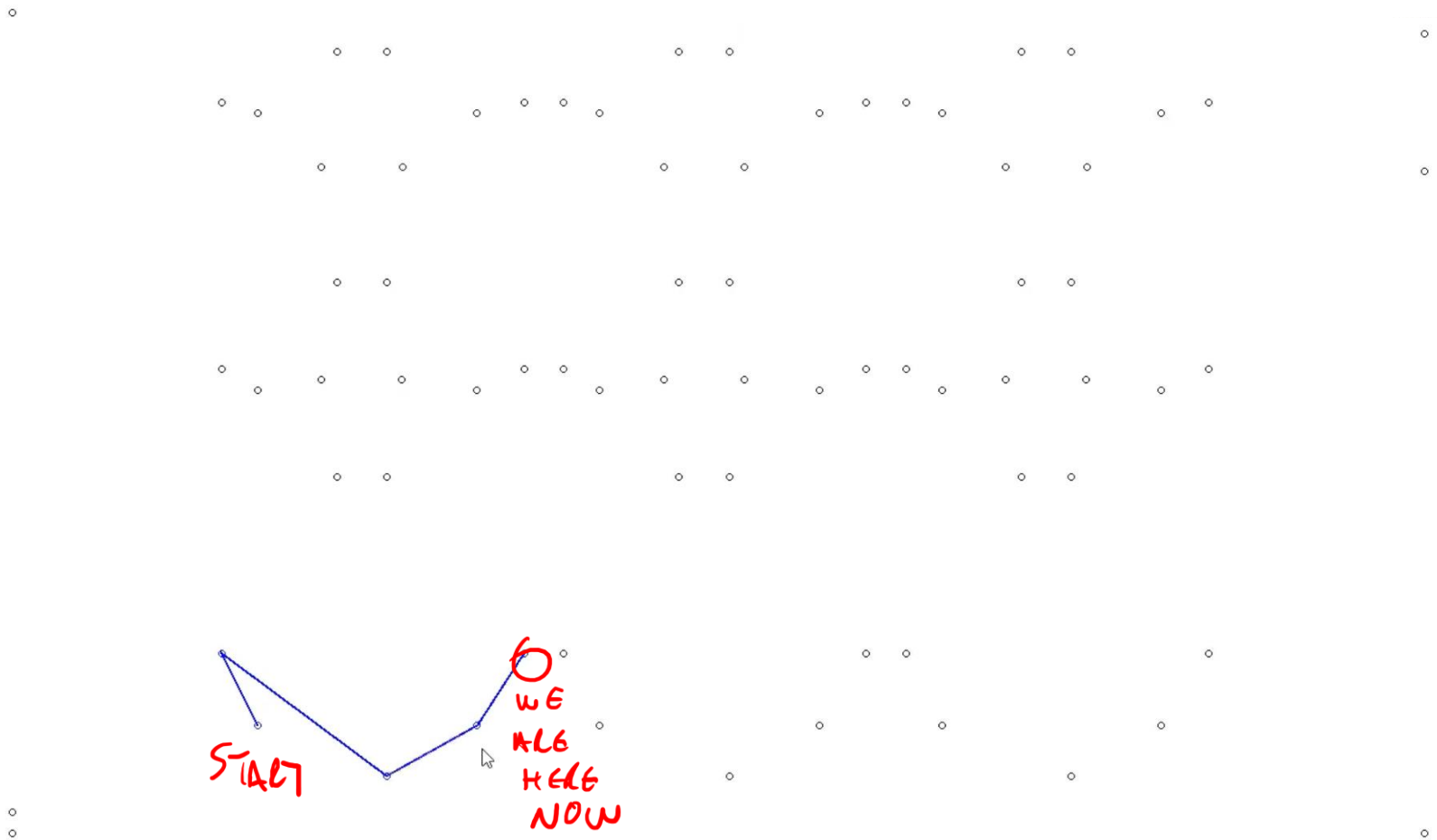
- For example: TSP
  - Starting with an empty tour, start in the “home” city
  - Repeat:
    - Add the city closest to the last-added city



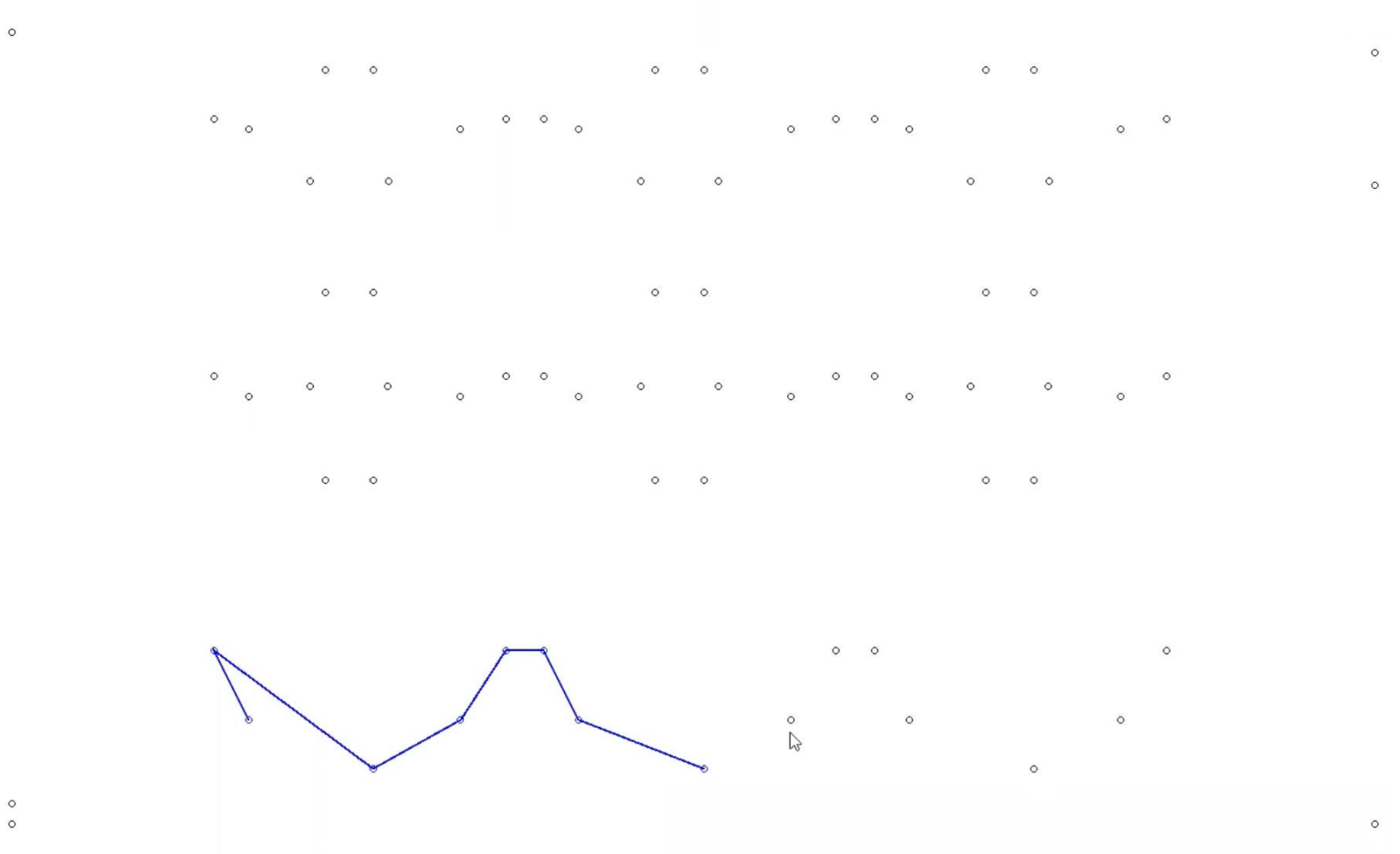
Construction algorithms – Greedy



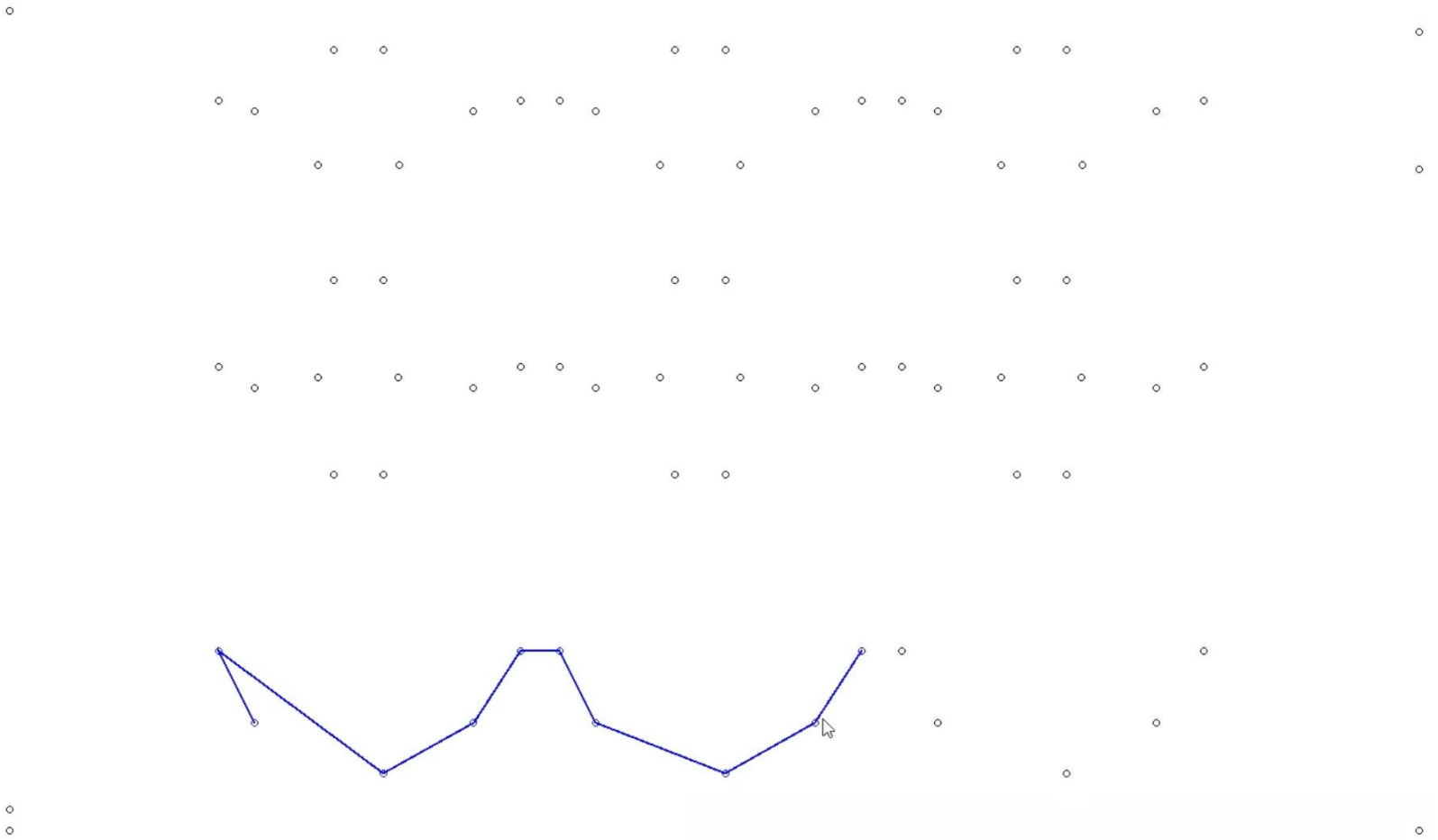
Construction algorithms – Greedy



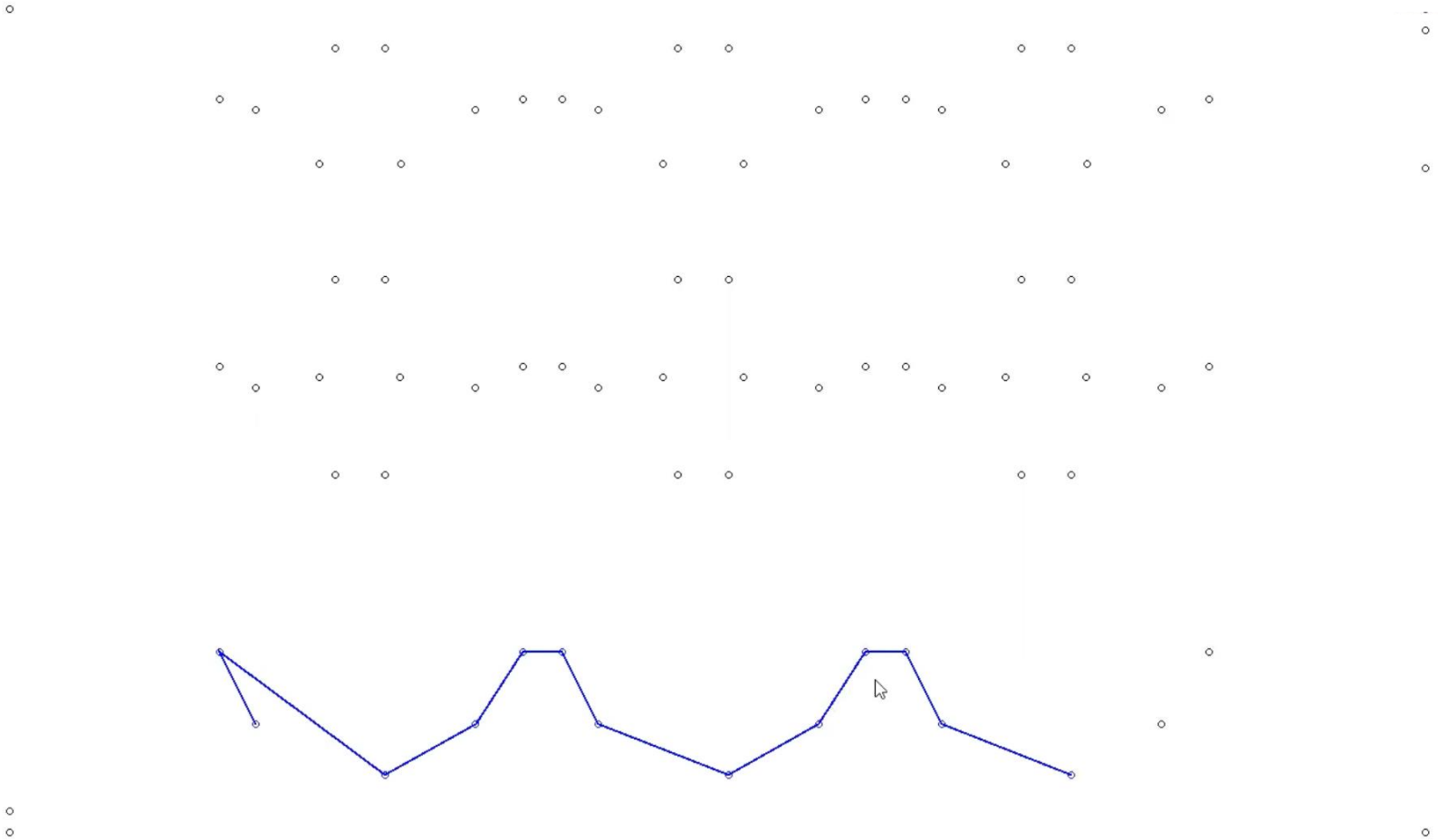
Construction algorithms – Greedy



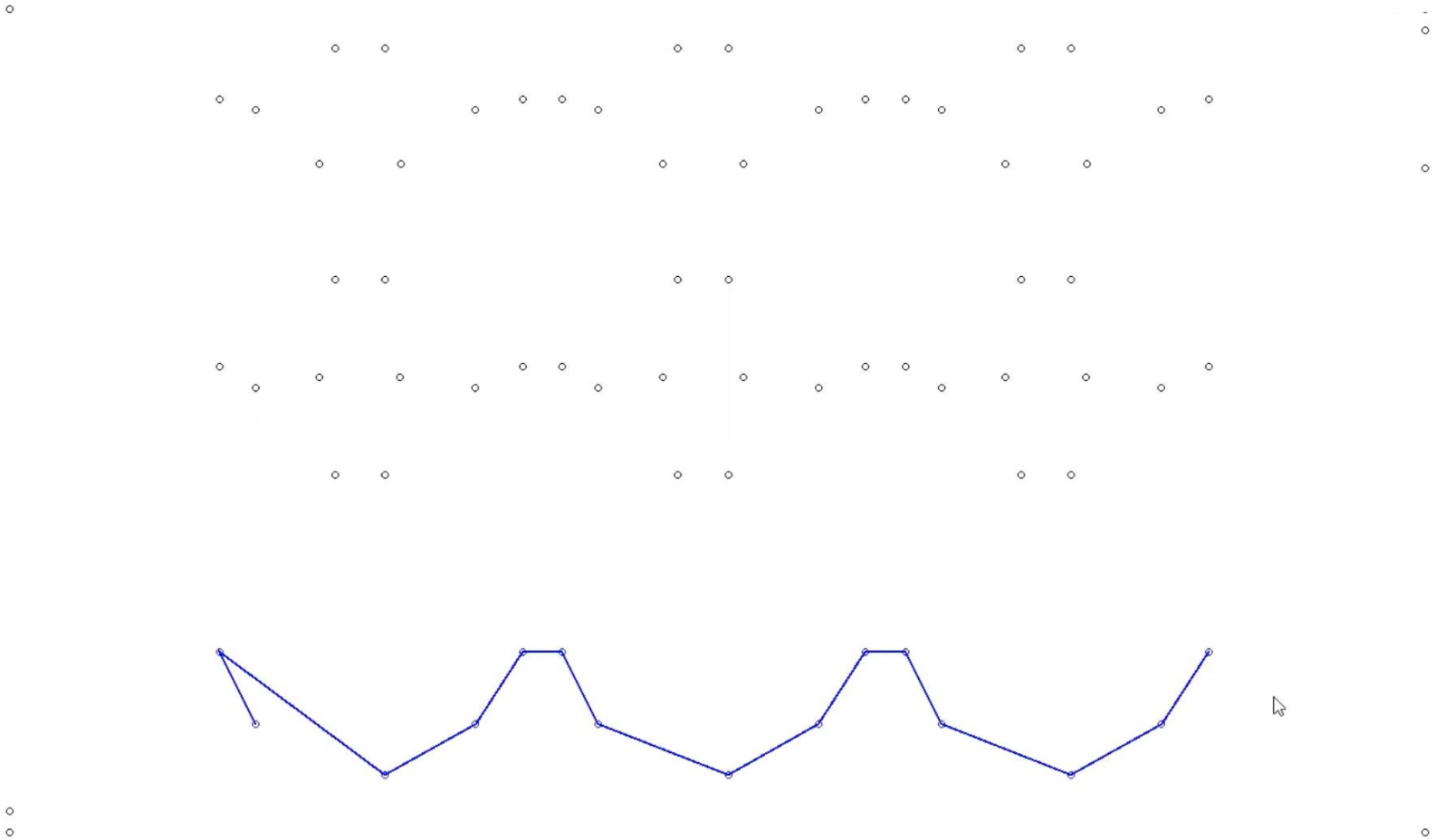
Construction algorithms – Greedy



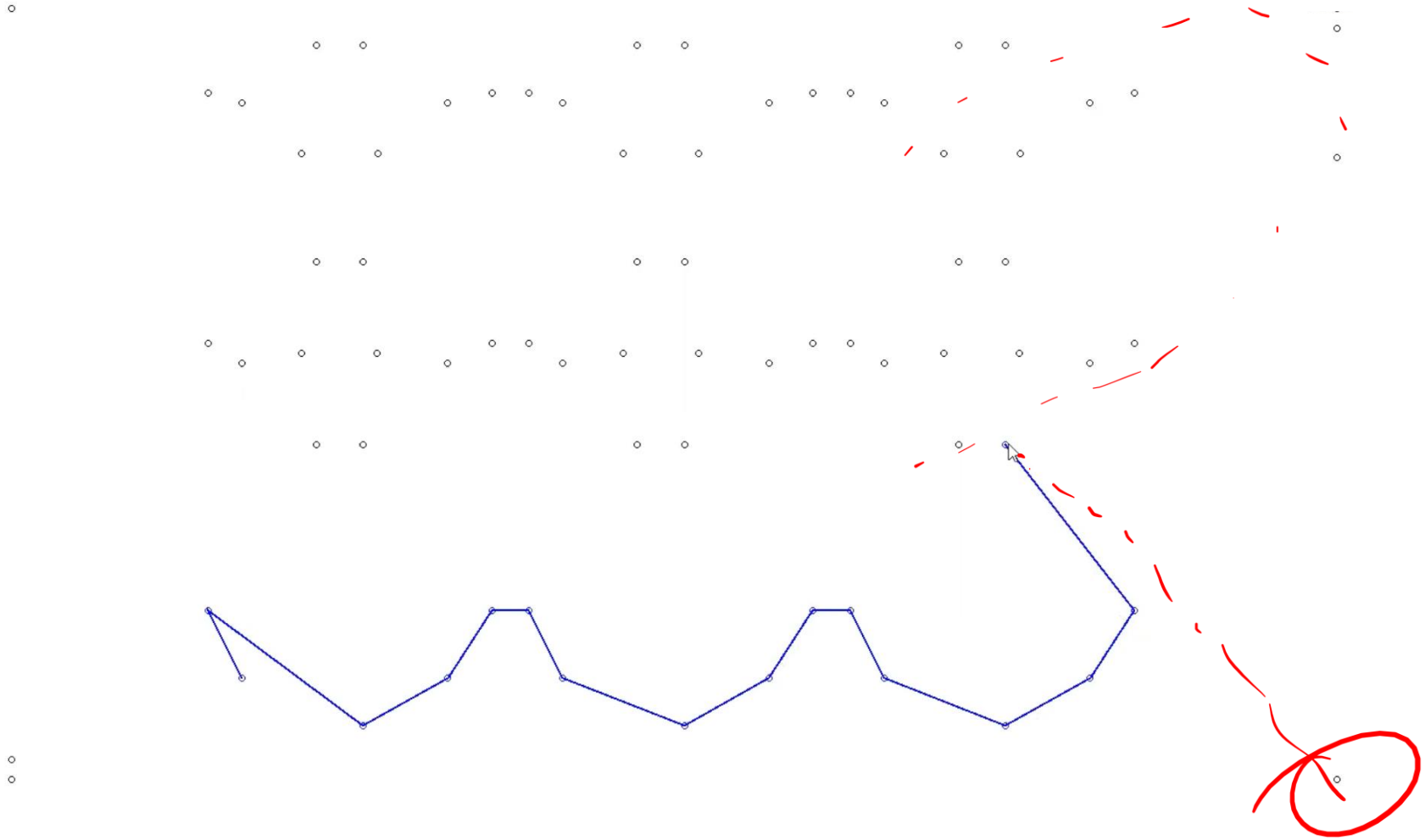
Construction algorithms – Greedy



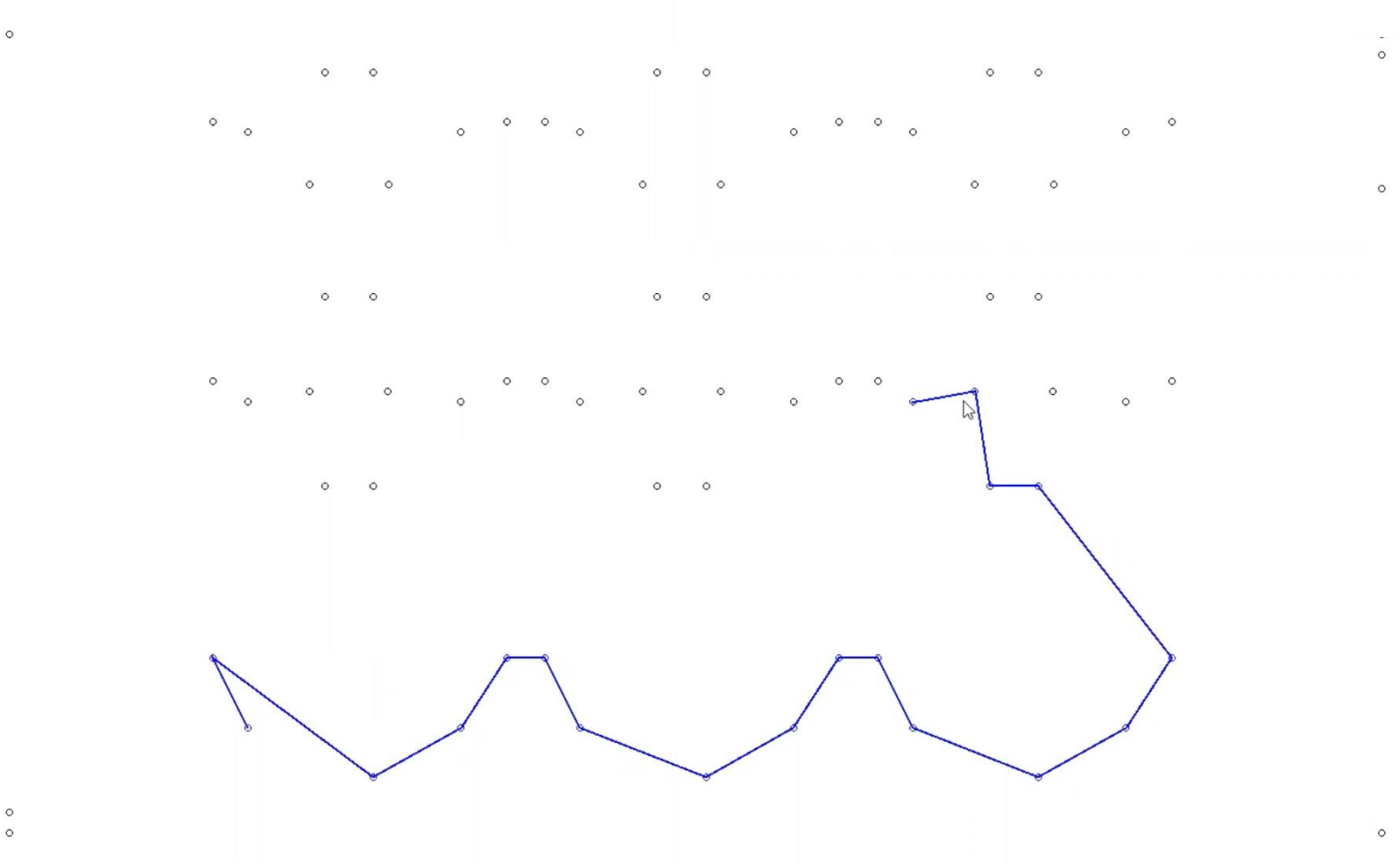
Construction algorithms – Greedy



Construction algorithms – Greedy



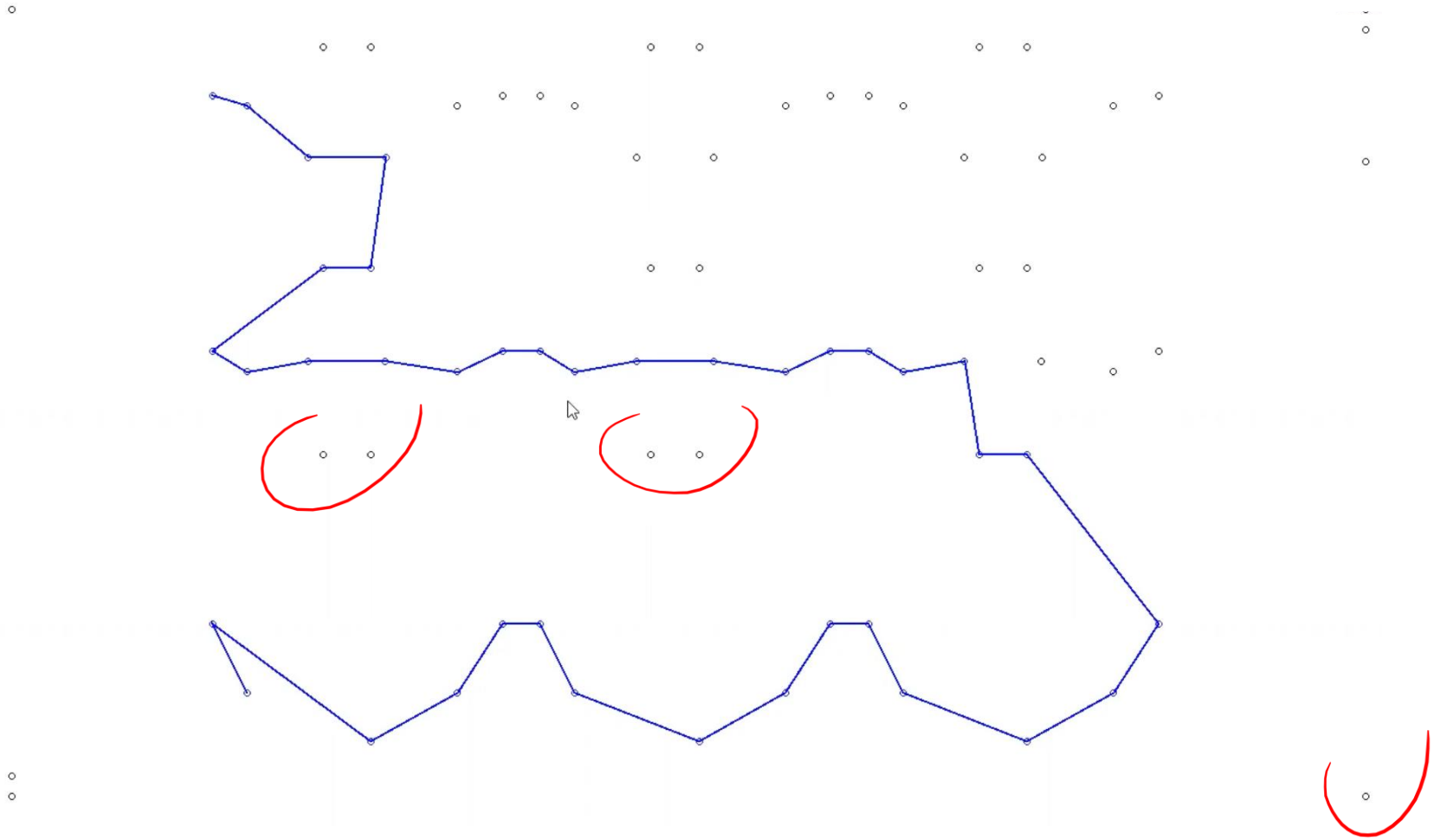
Construction algorithms – Greedy



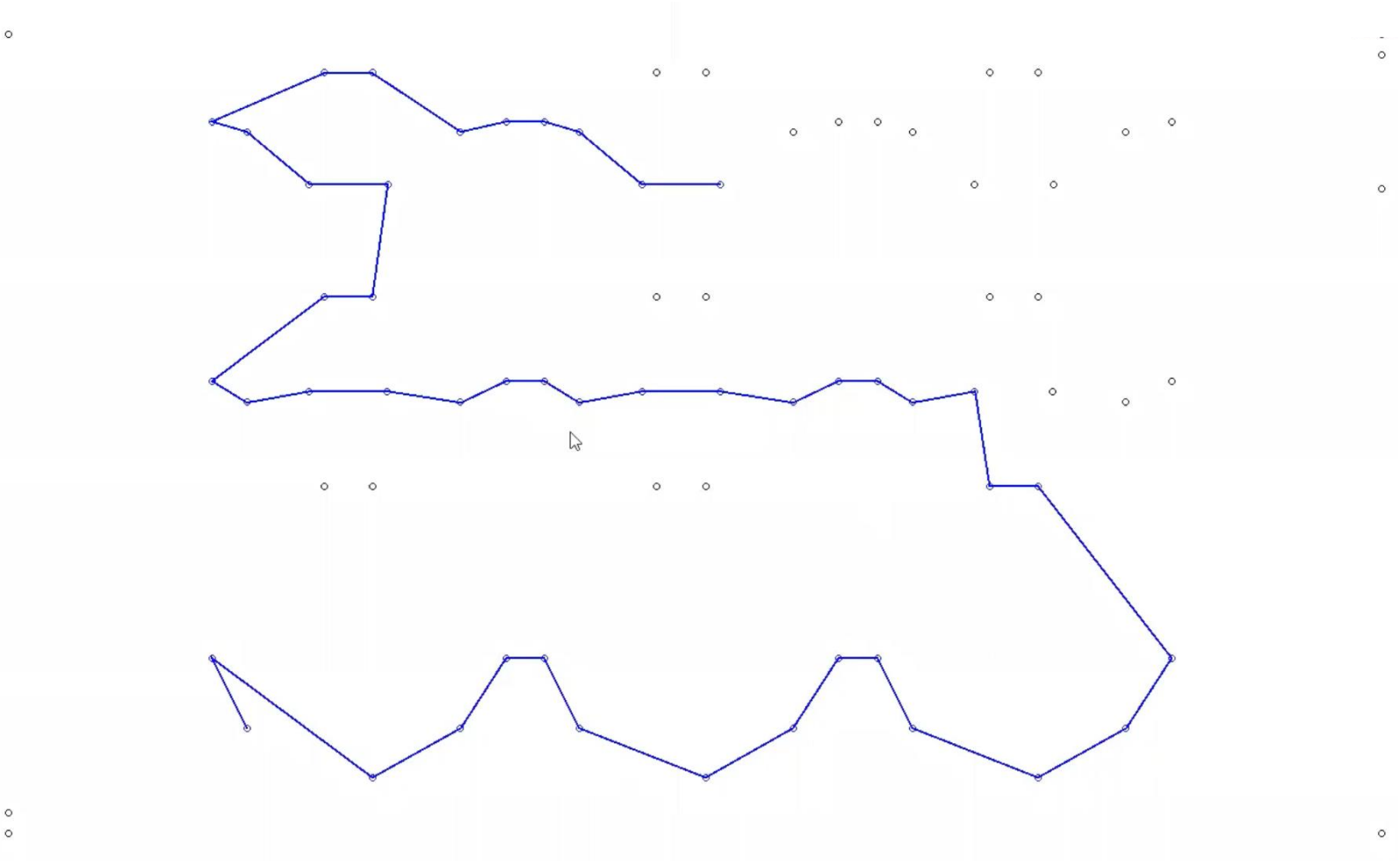
Construction algorithms – Greedy

Construction algorithms – Greedy

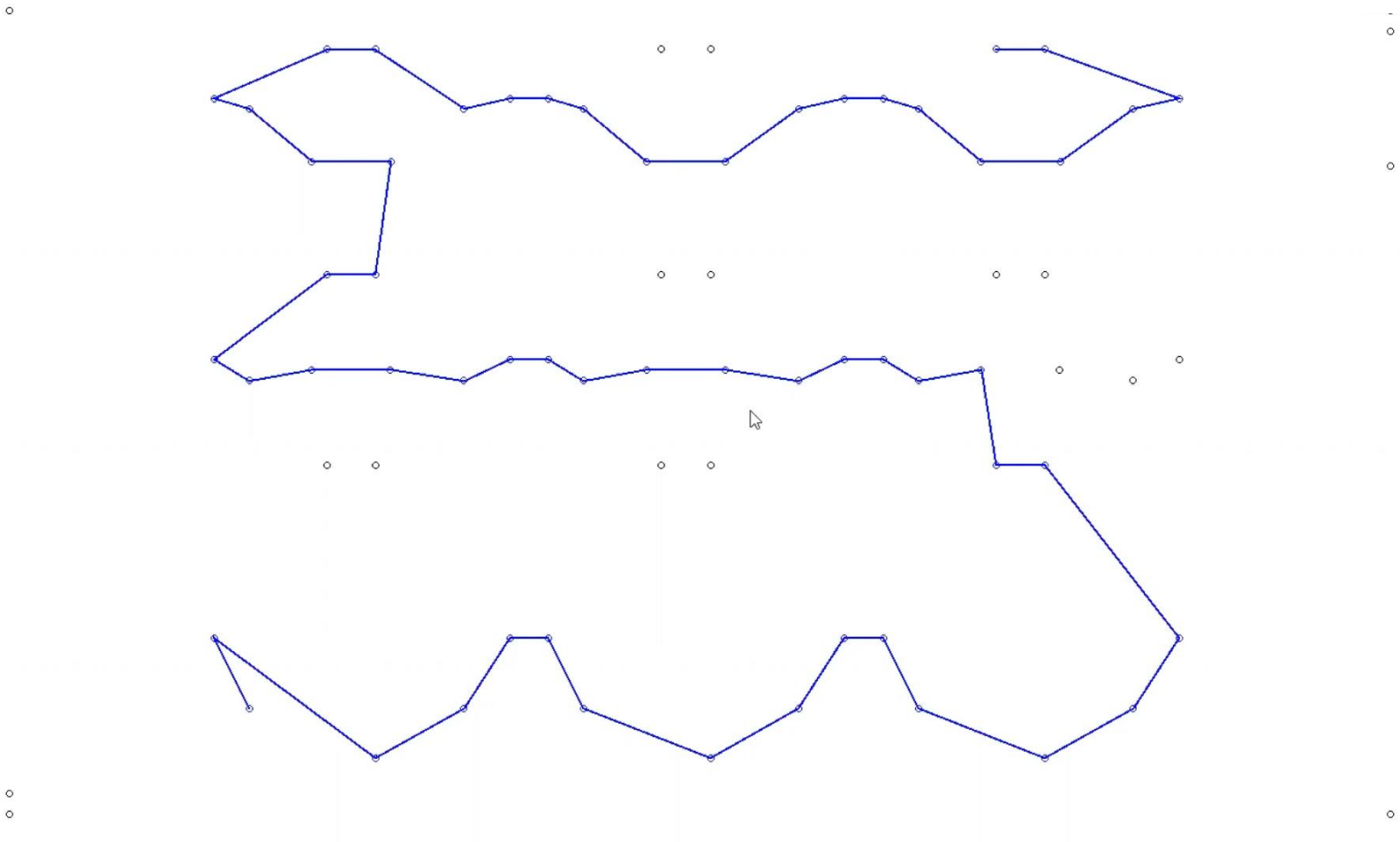
Construction algorithms – Greedy



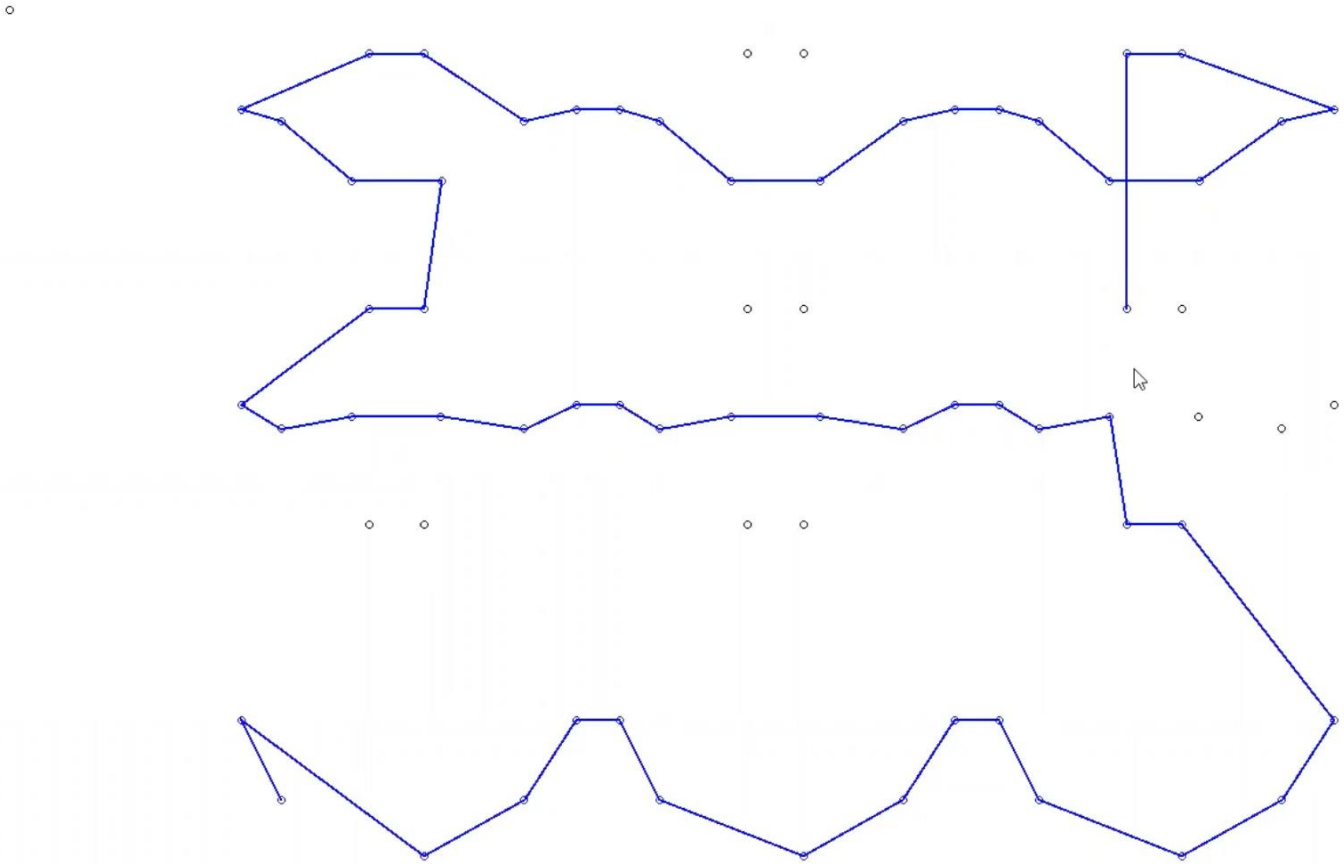
Construction algorithms – Greedy



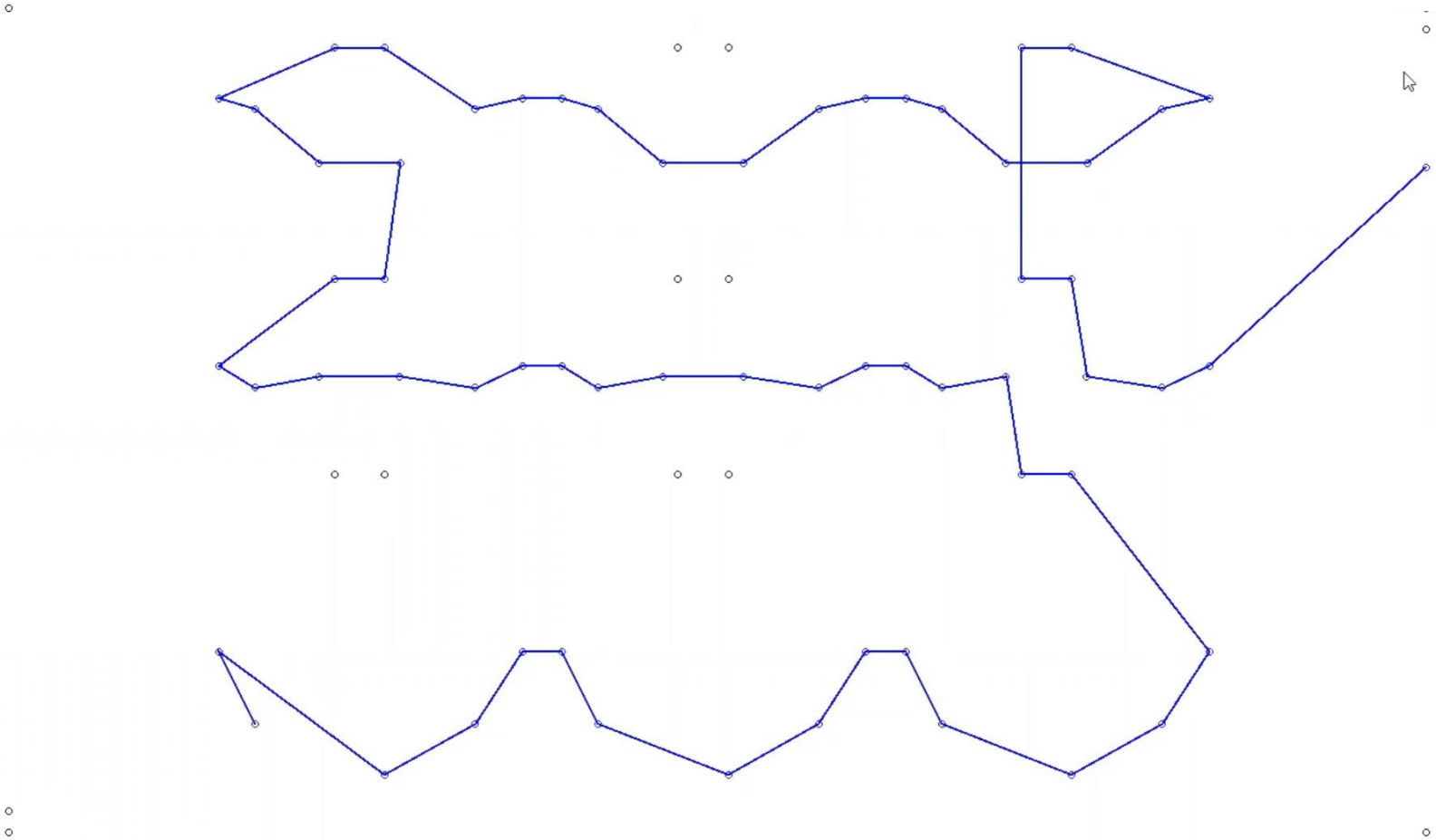
Construction algorithms – Greedy



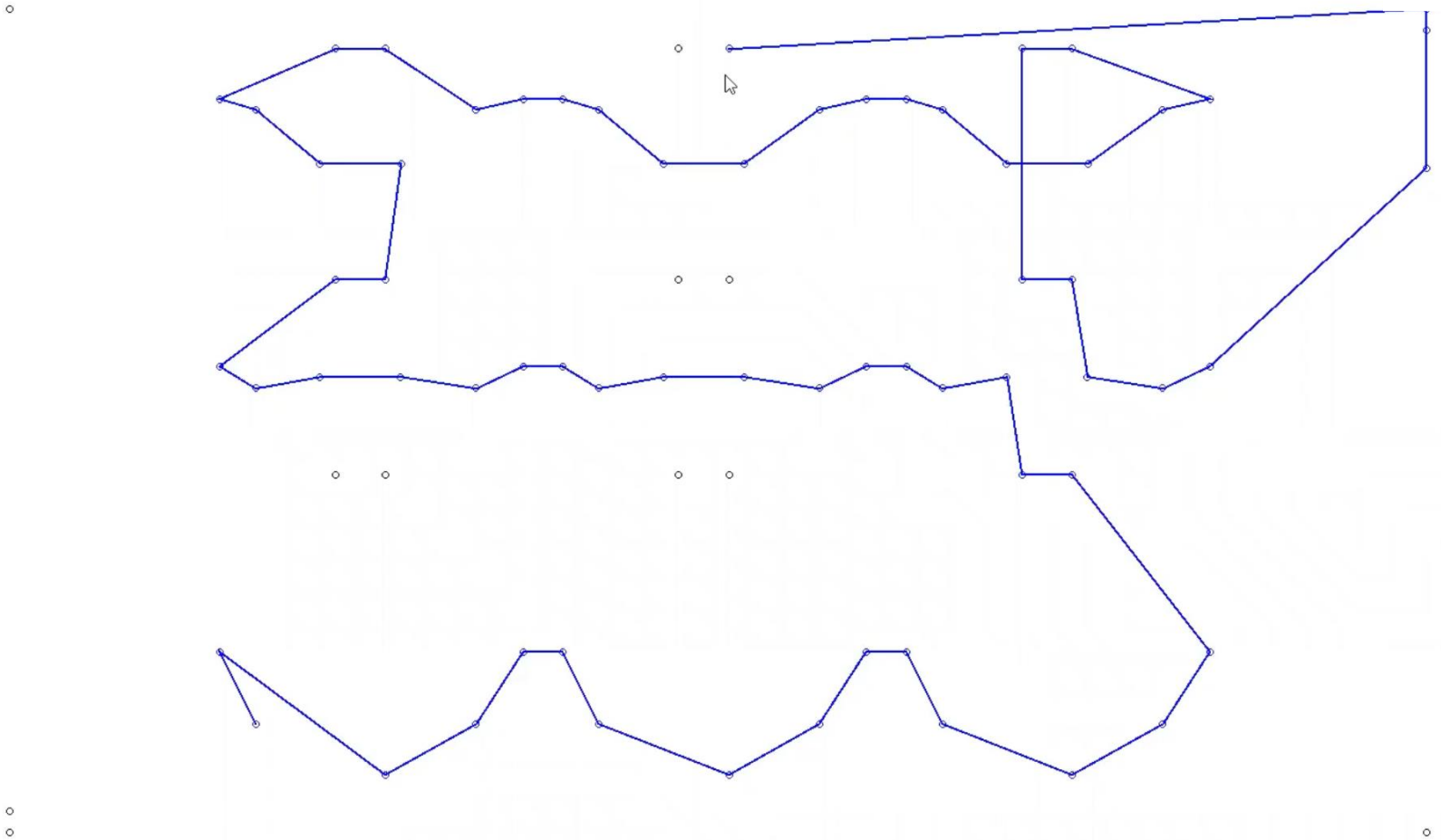
Construction algorithms – Greedy



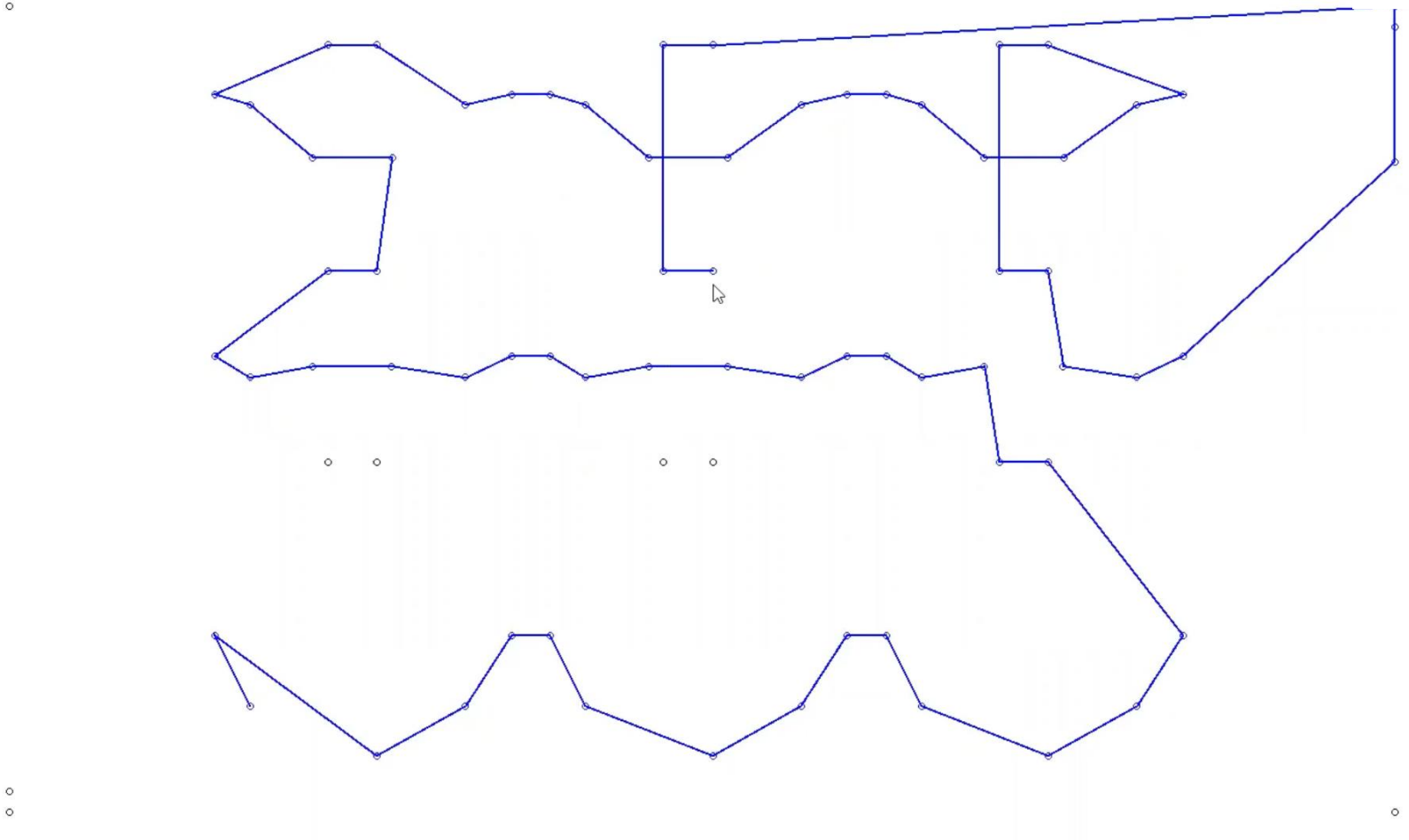
Construction algorithms – Greedy



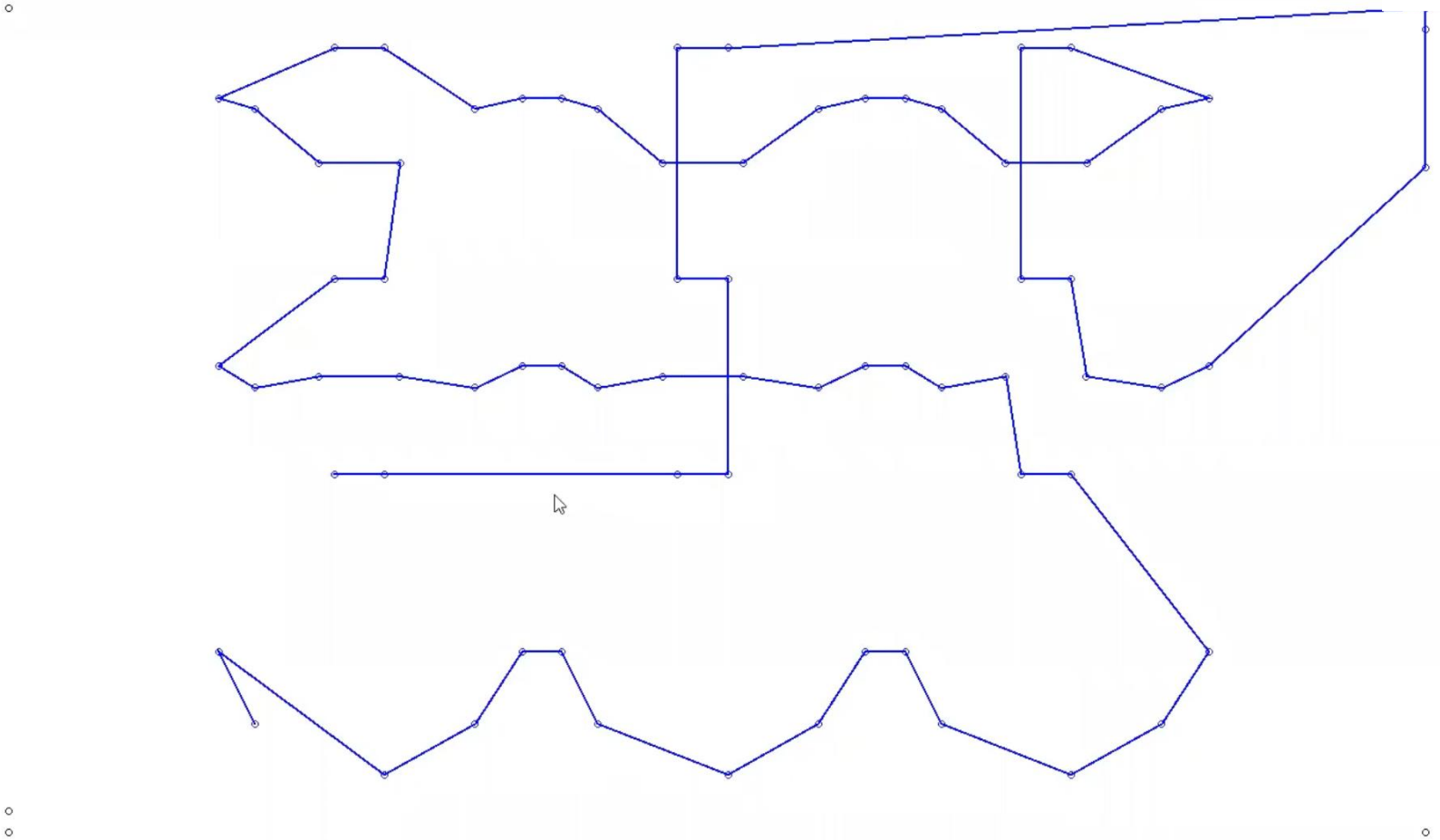
Construction algorithms – Greedy



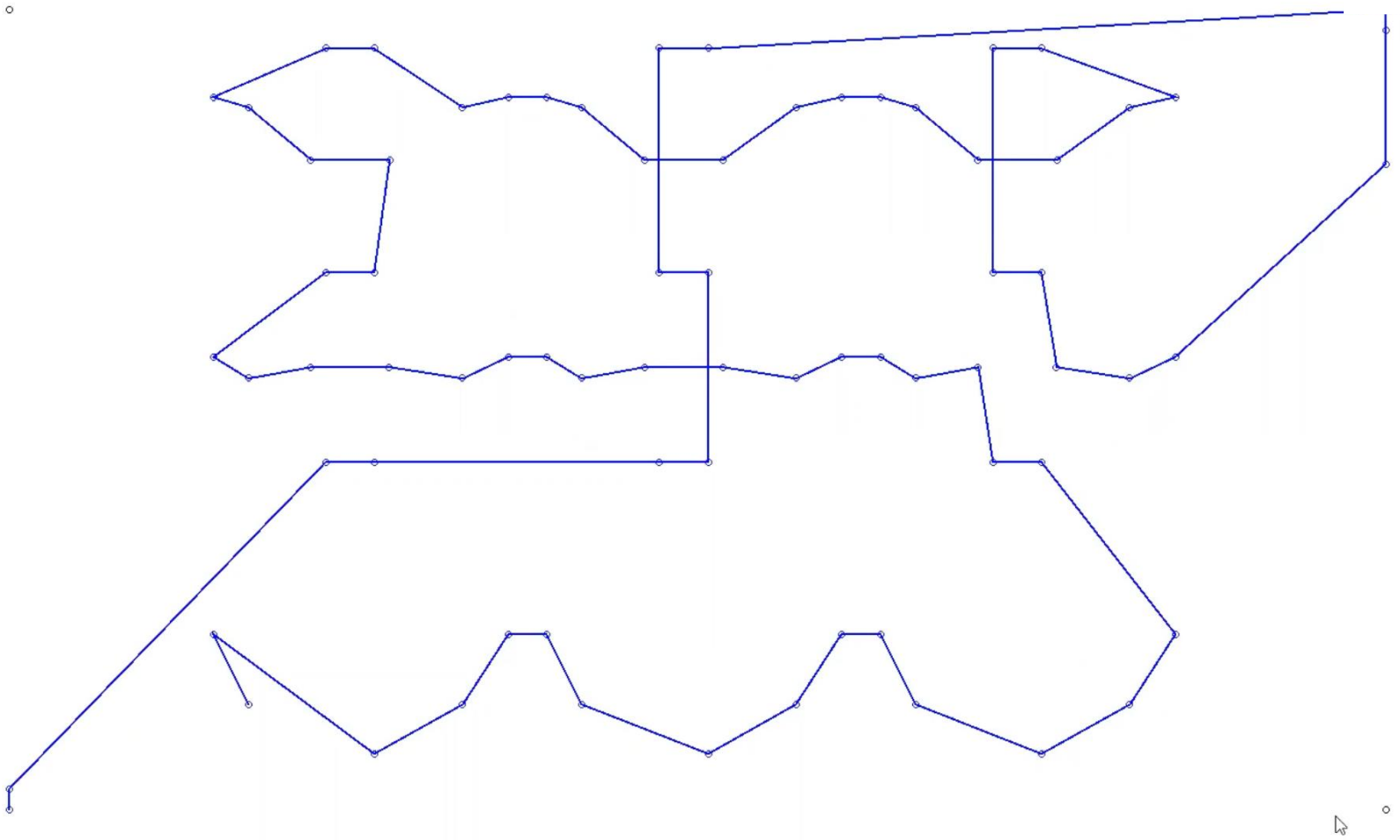
Construction algorithms – Greedy



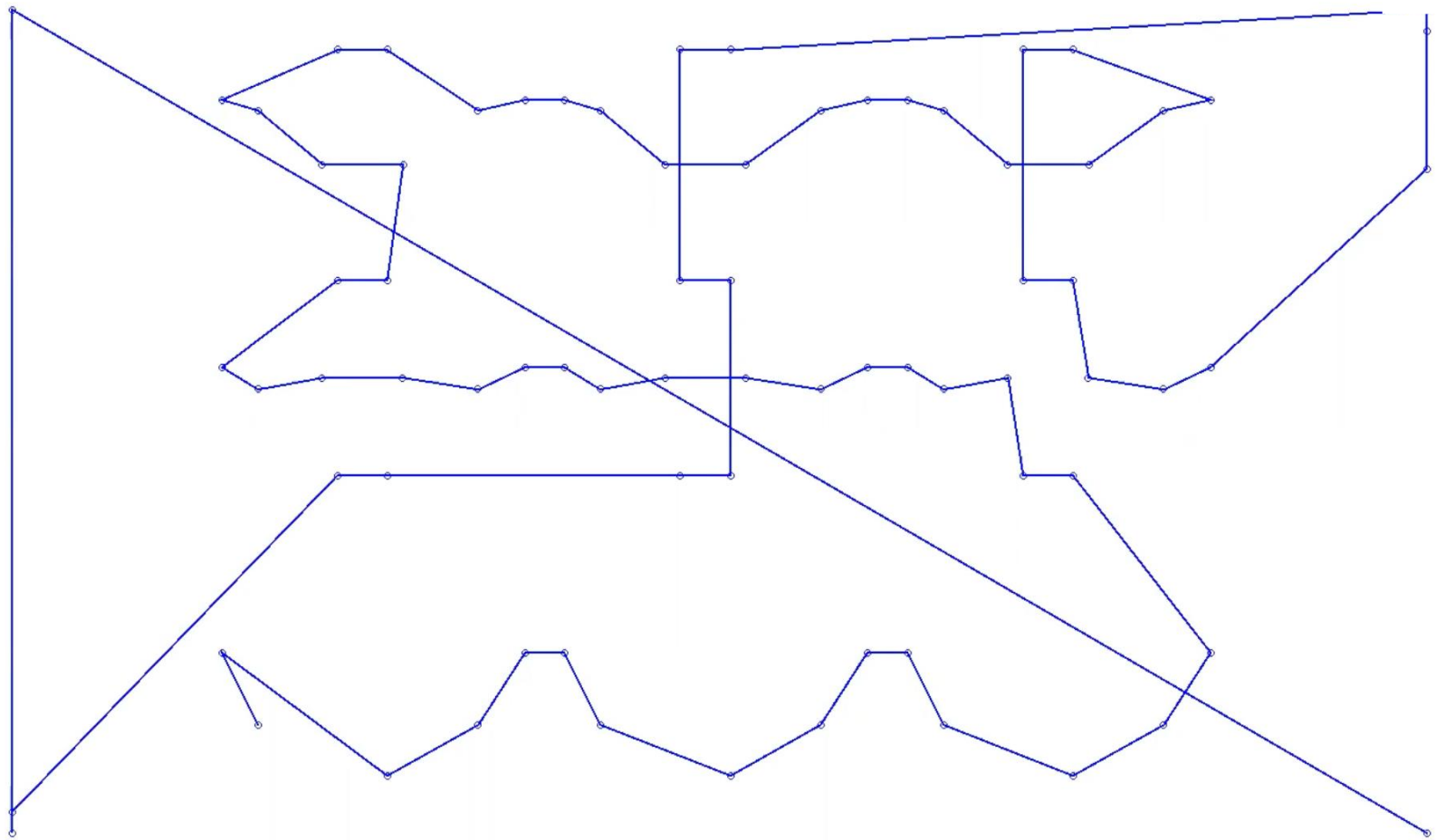
Construction algorithms – Greedy



Construction algorithms – Greedy



Construction algorithms – Greedy



Construction algorithms – Greedy

- "Nearest Neighbour" is a classic greedy algorithm
- Greedy algorithms perform locally-optimal moves
- However, they often make "structural" errors.

- Greedy algorithms exist for many problems:
  - Knapsack: Insert the item with the best value/weight ratio
  - Scheduling: Schedule the item with the earliest deadline

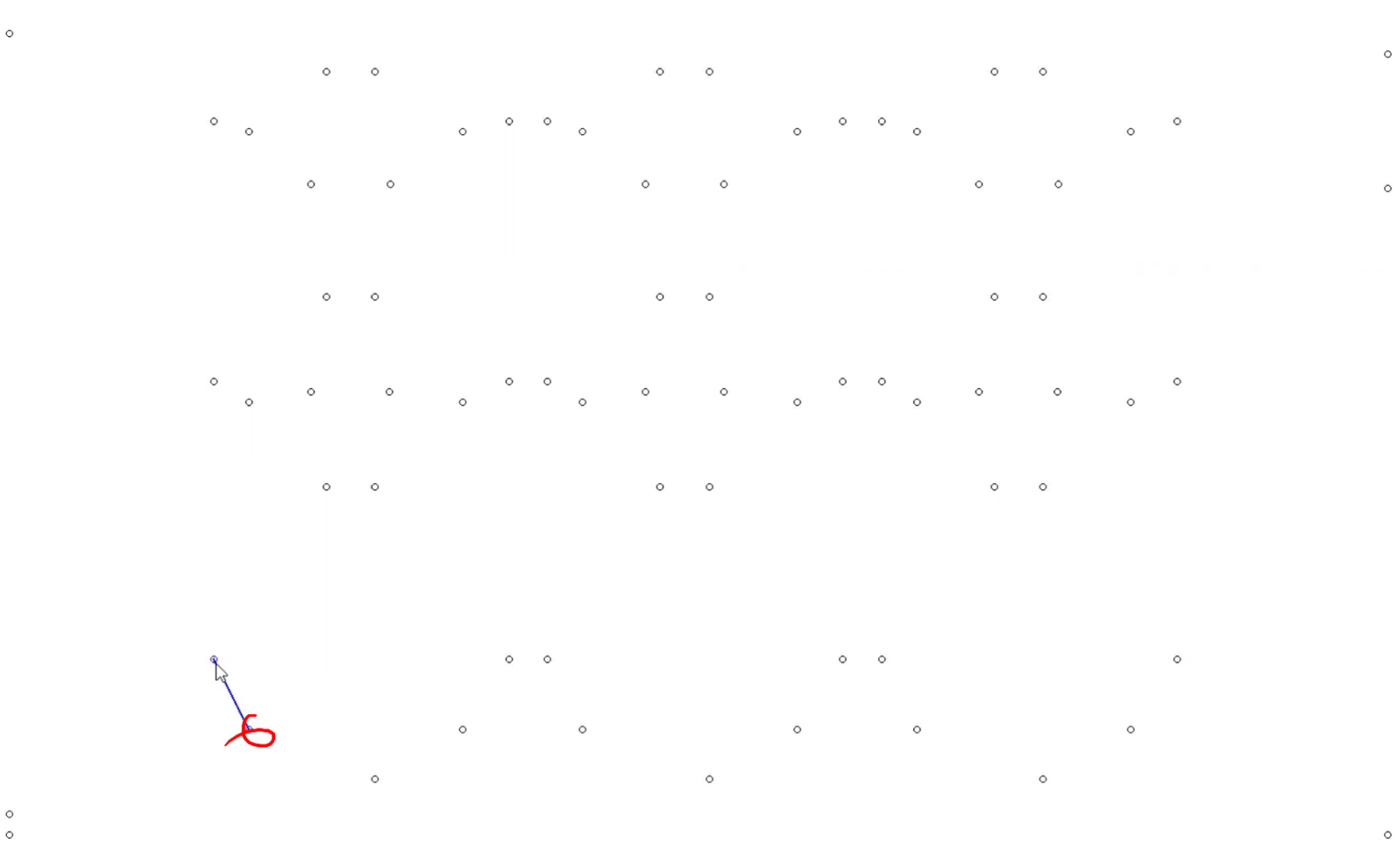
Construction algorithms – Minimum Insert Cost

A better greedy algorithm

- "Minimum Insert" is another greedy algorithm – but performs slightly better

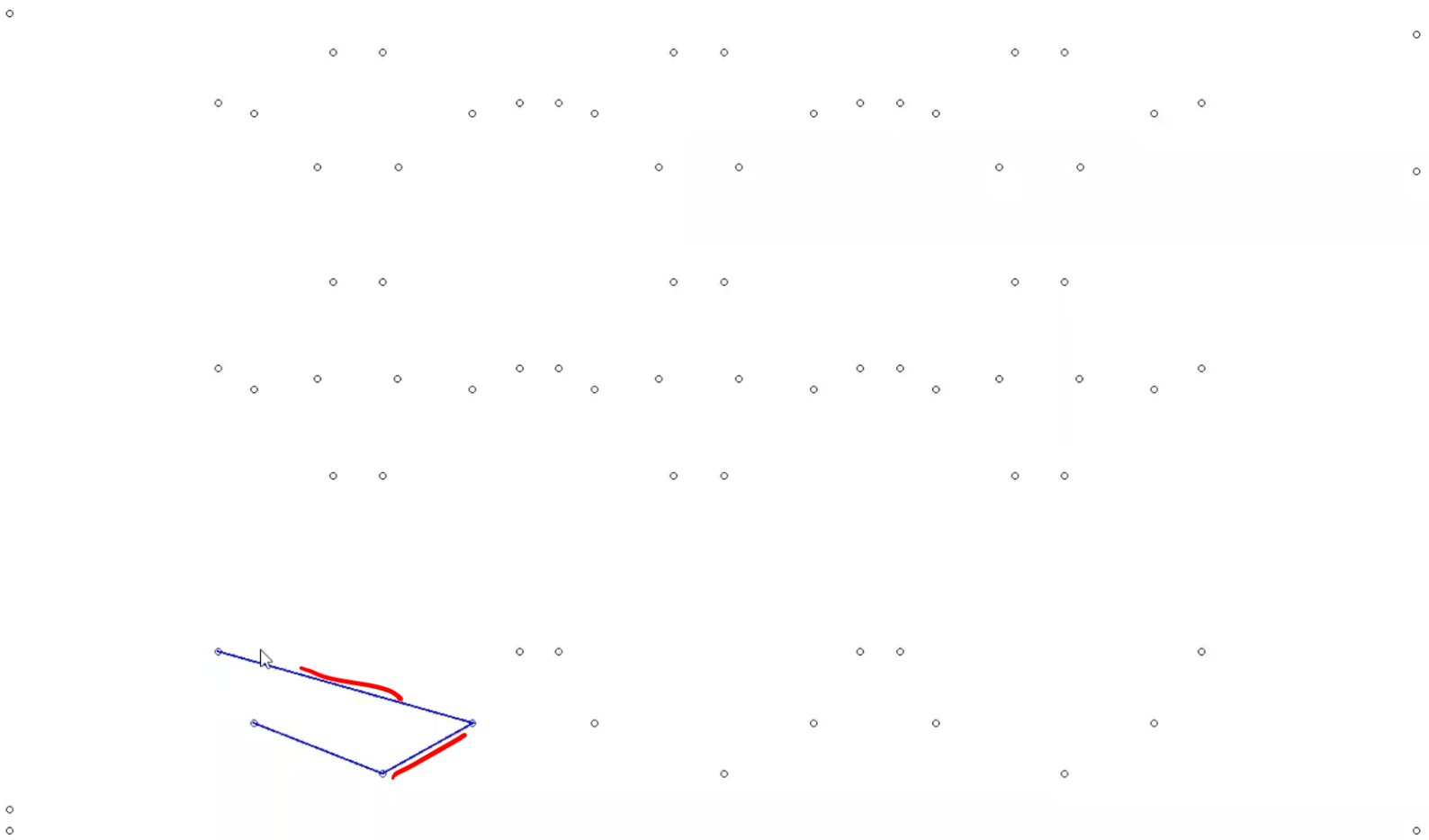
- At each iteration
  - Find the customer that, when inserted, increases the cost by the least.
  - Insert it in the position that increases the cost by the least
  - Allows insert between *any* pair of customers – not just at the end

Construction algorithms – Minimum Insert Cost

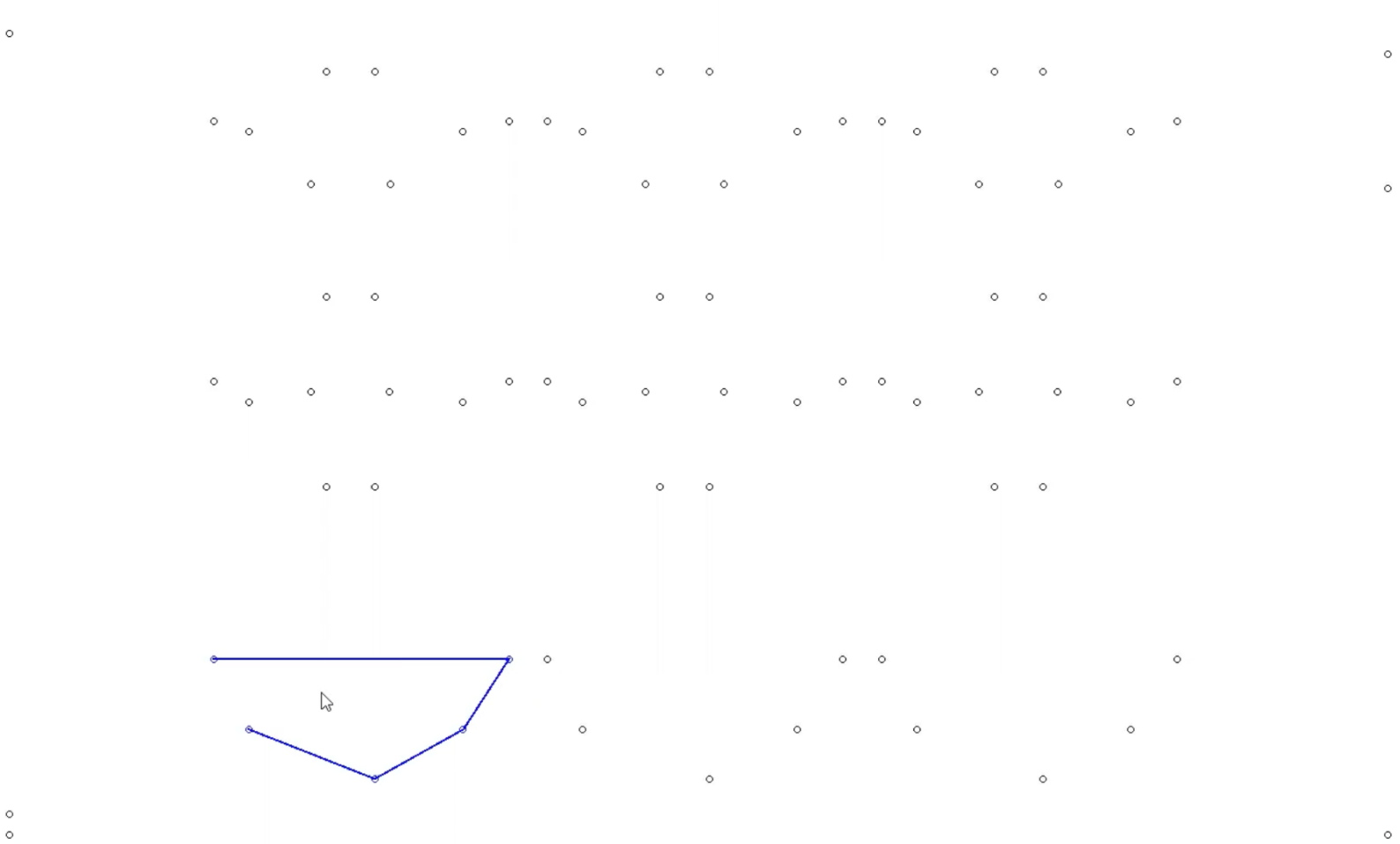


Construction algorithms – Minimum Insert Cost

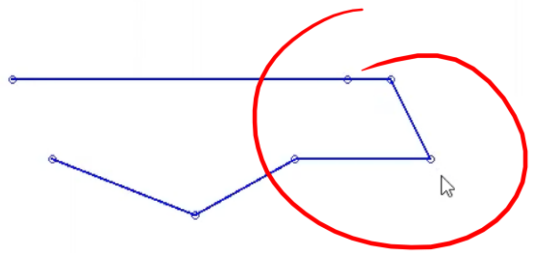
Construction algorithms – Minimum Insert Cost



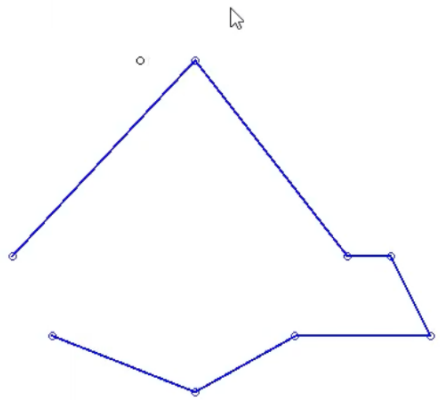
Construction algorithms – Minimum Insert Cost



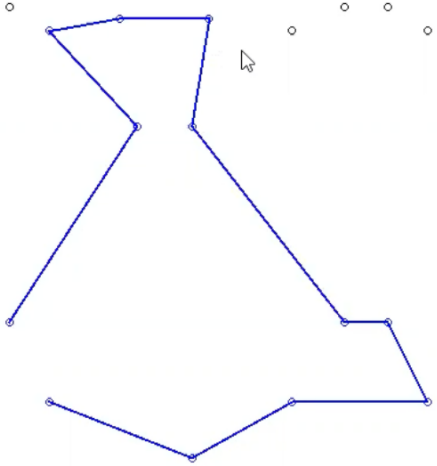
Construction algorithms – Minimum Insert Cost



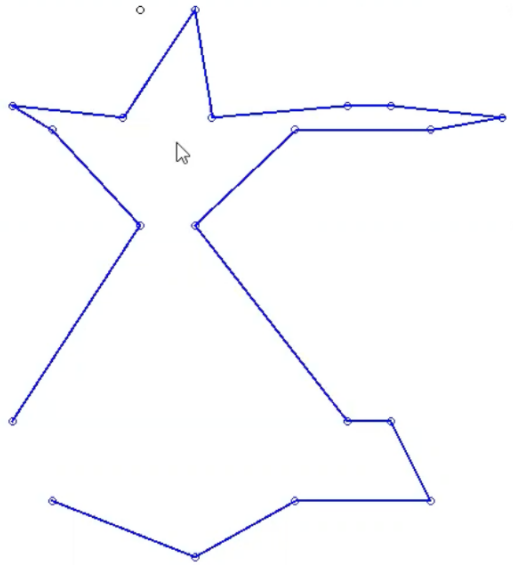
Construction algorithms – Minimum Insert Cost



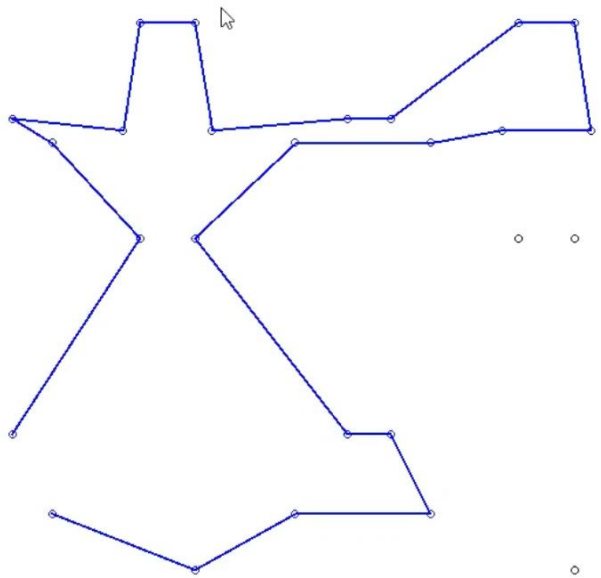
Construction algorithms – Minimum Insert Cost



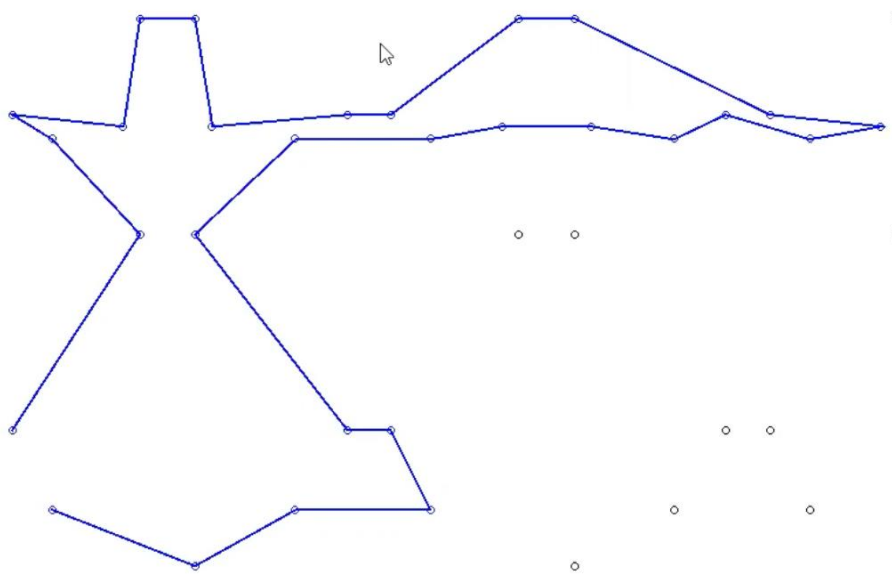
Construction algorithms – Minimum Insert Cost



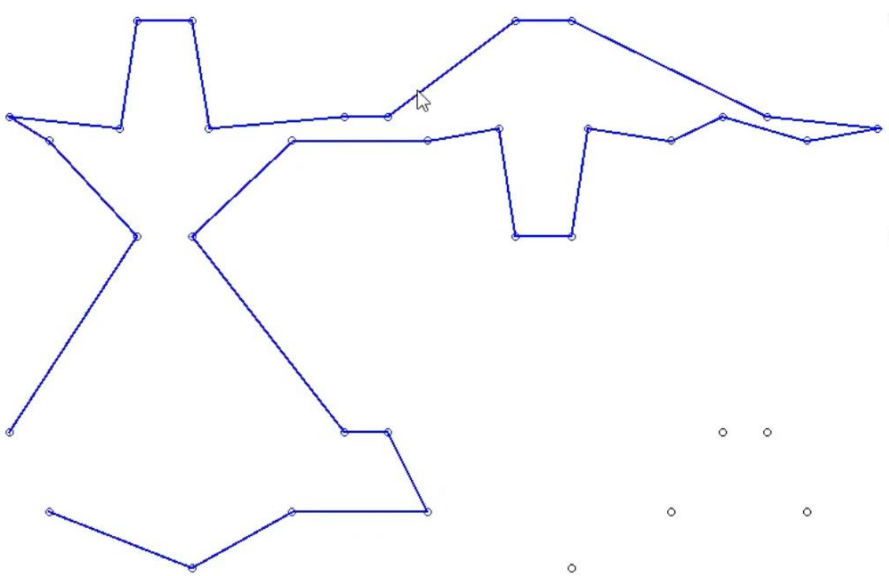
Construction algorithms – Minimum Insert Cost



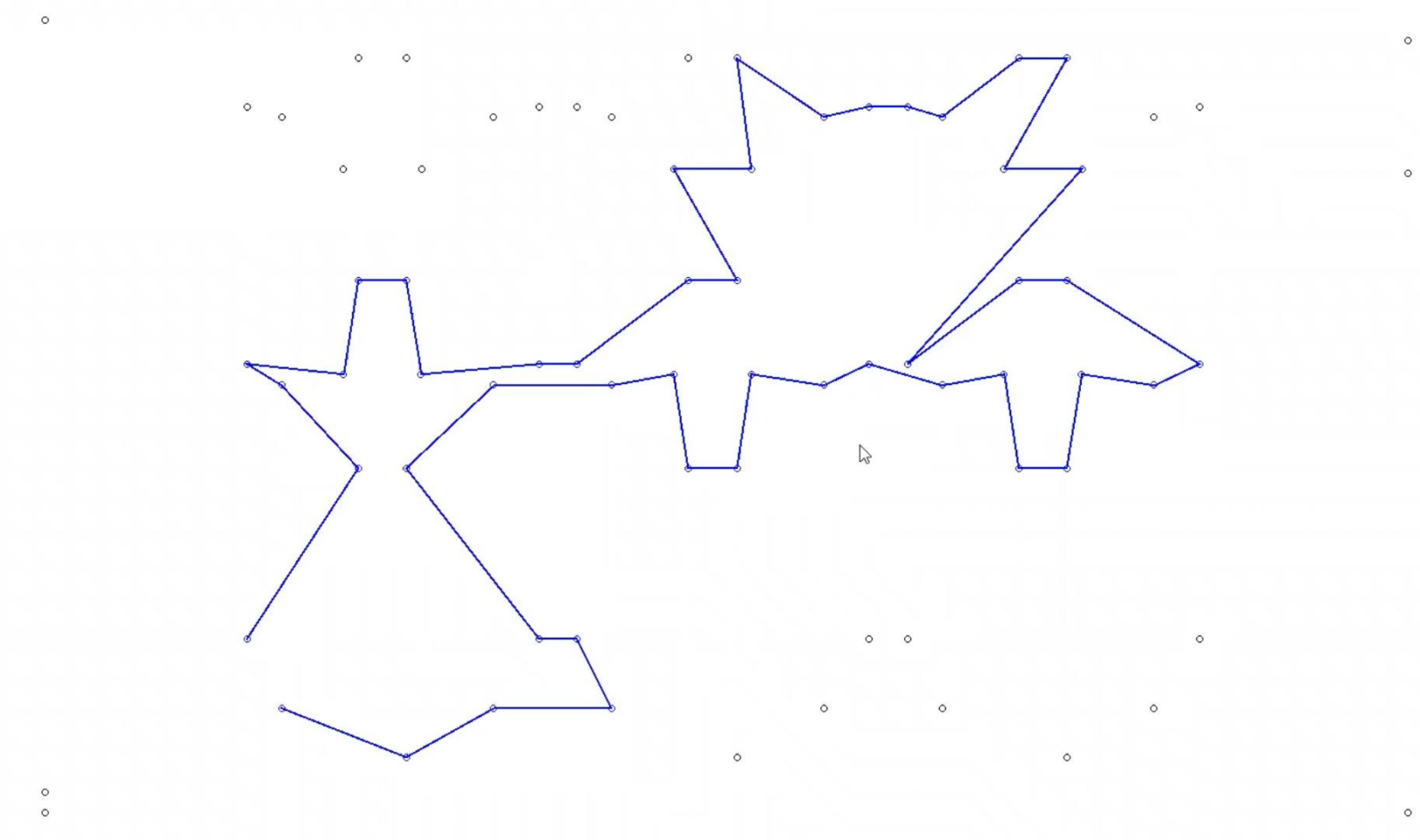
Construction algorithms – Minimum Insert Cost



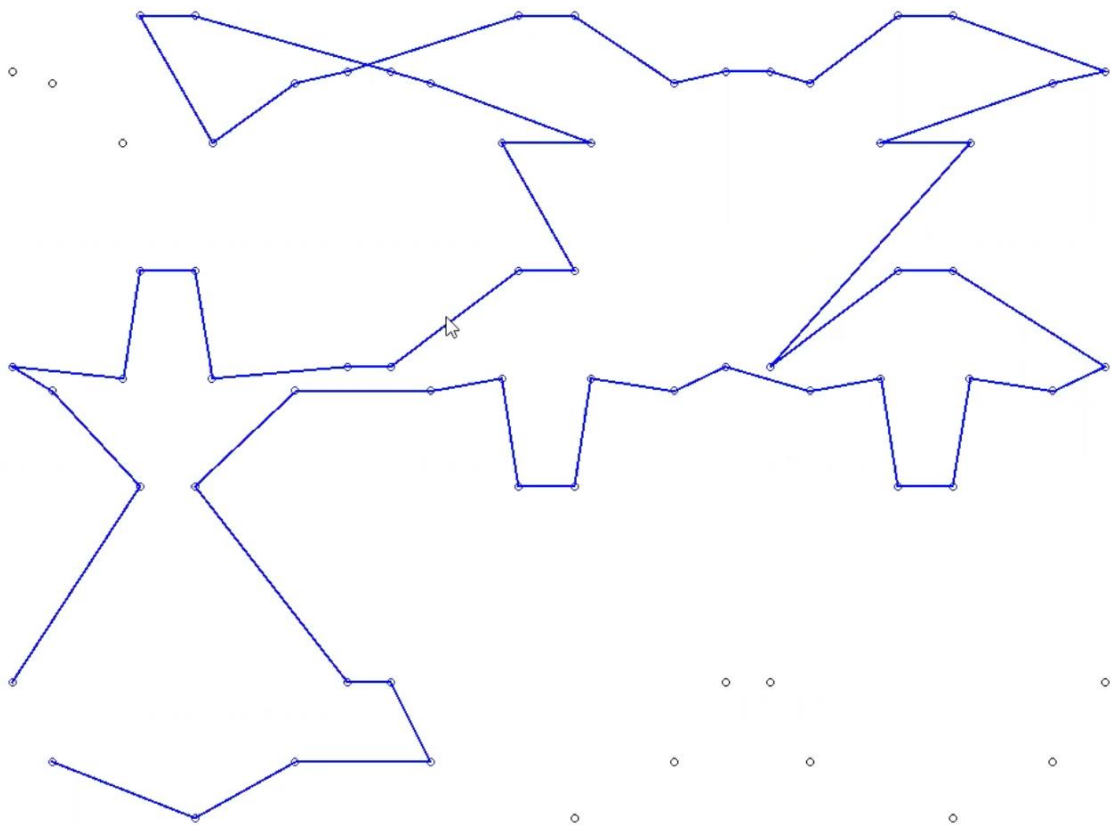
Construction algorithms – Minimum Insert Cost



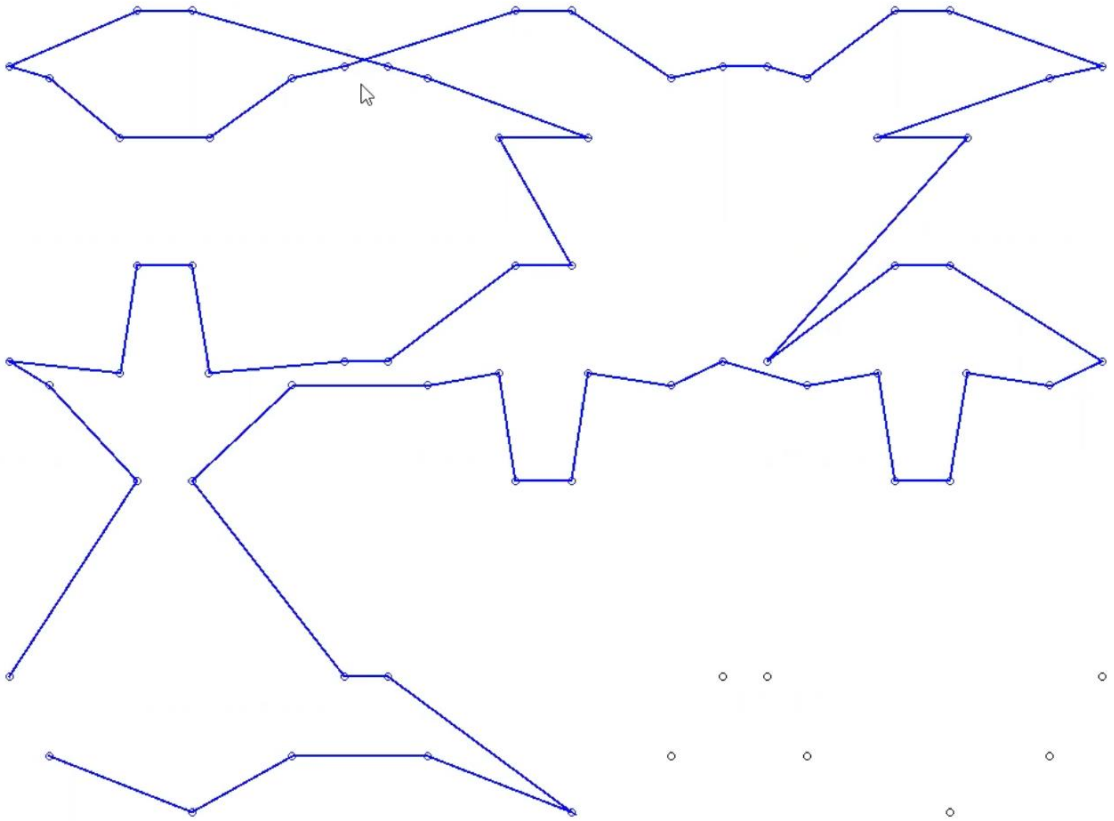
Construction algorithms – Minimum Insert Cost



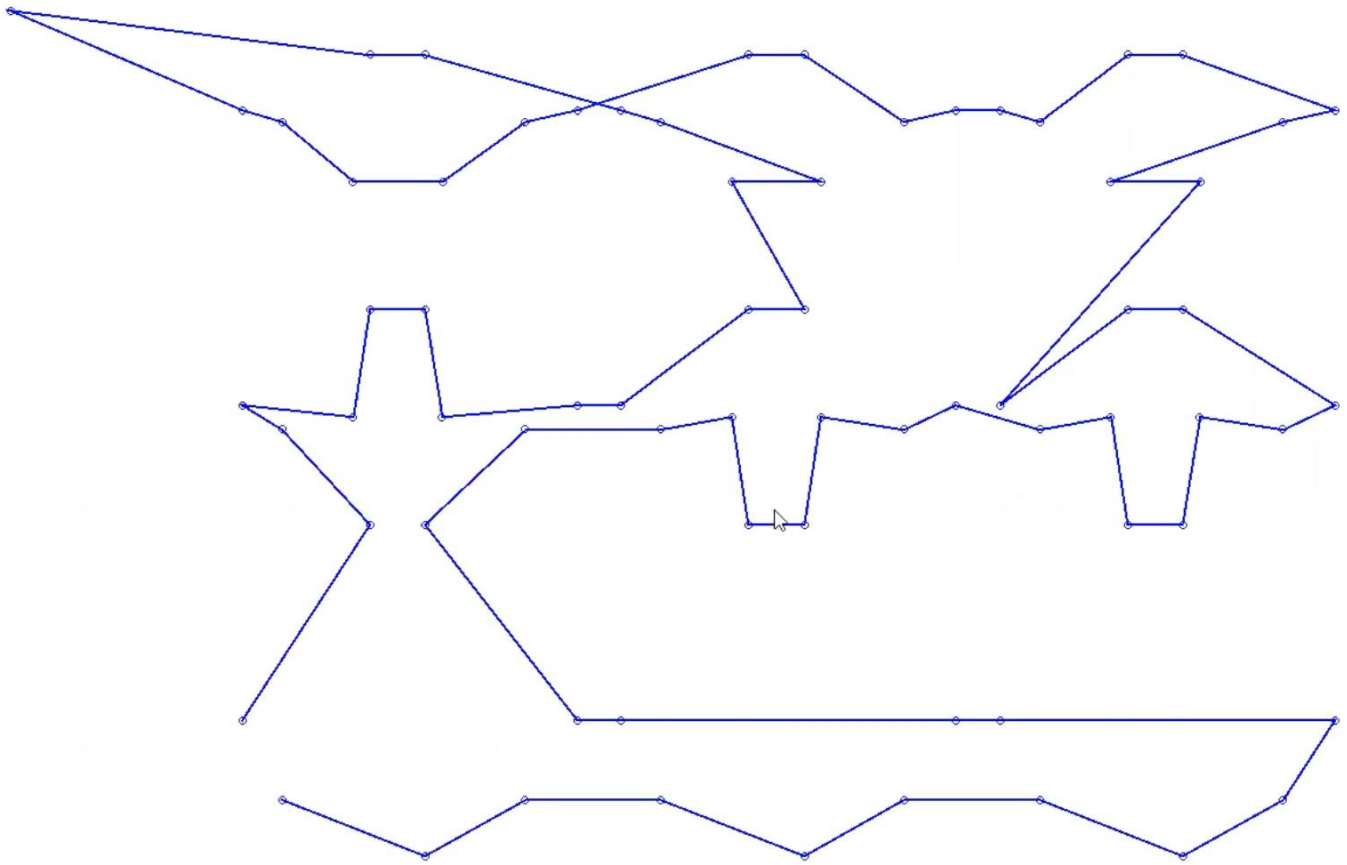
Construction algorithms – Minimum Insert Cost



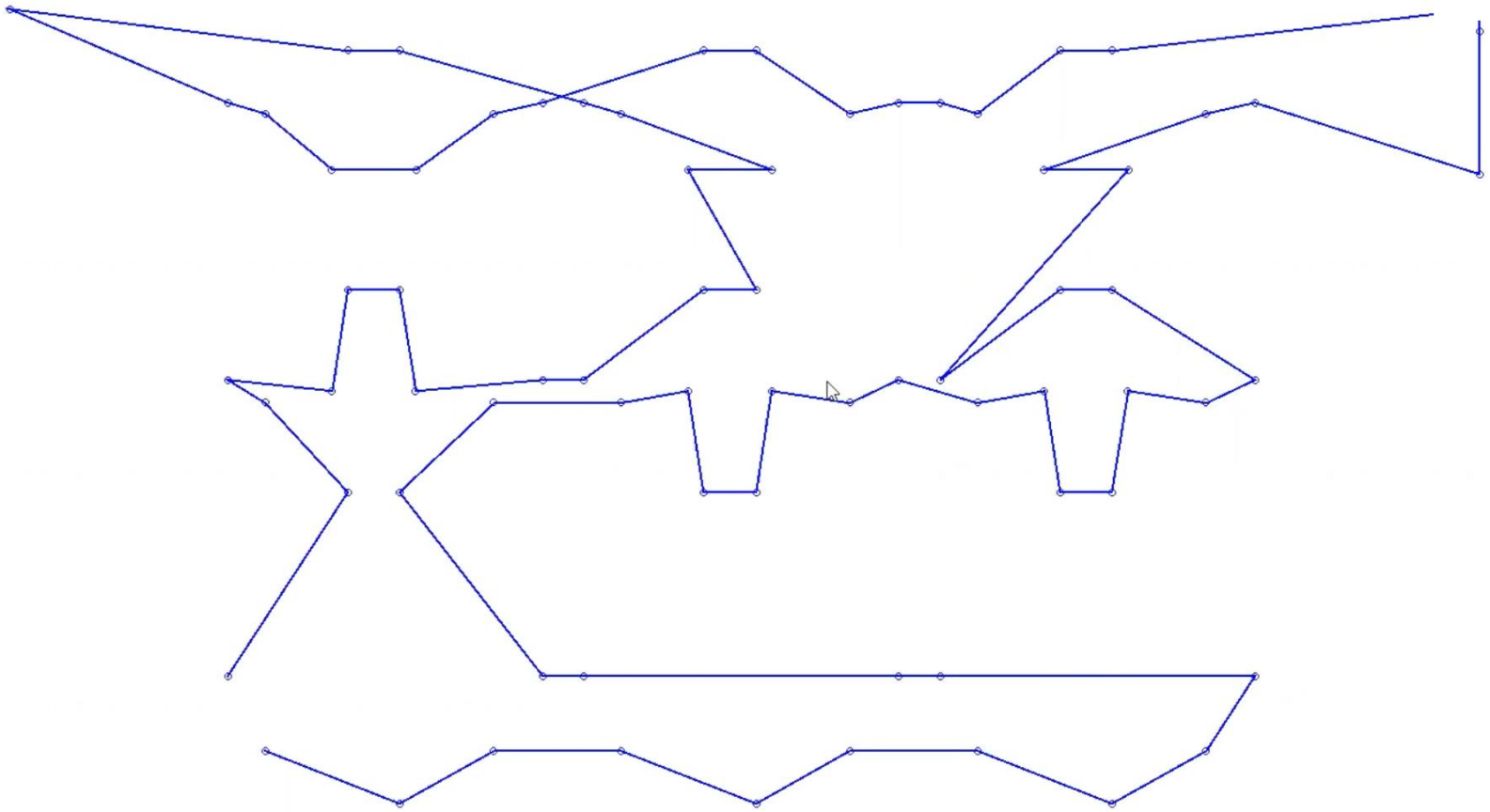
Construction algorithms – Minimum Insert Cost



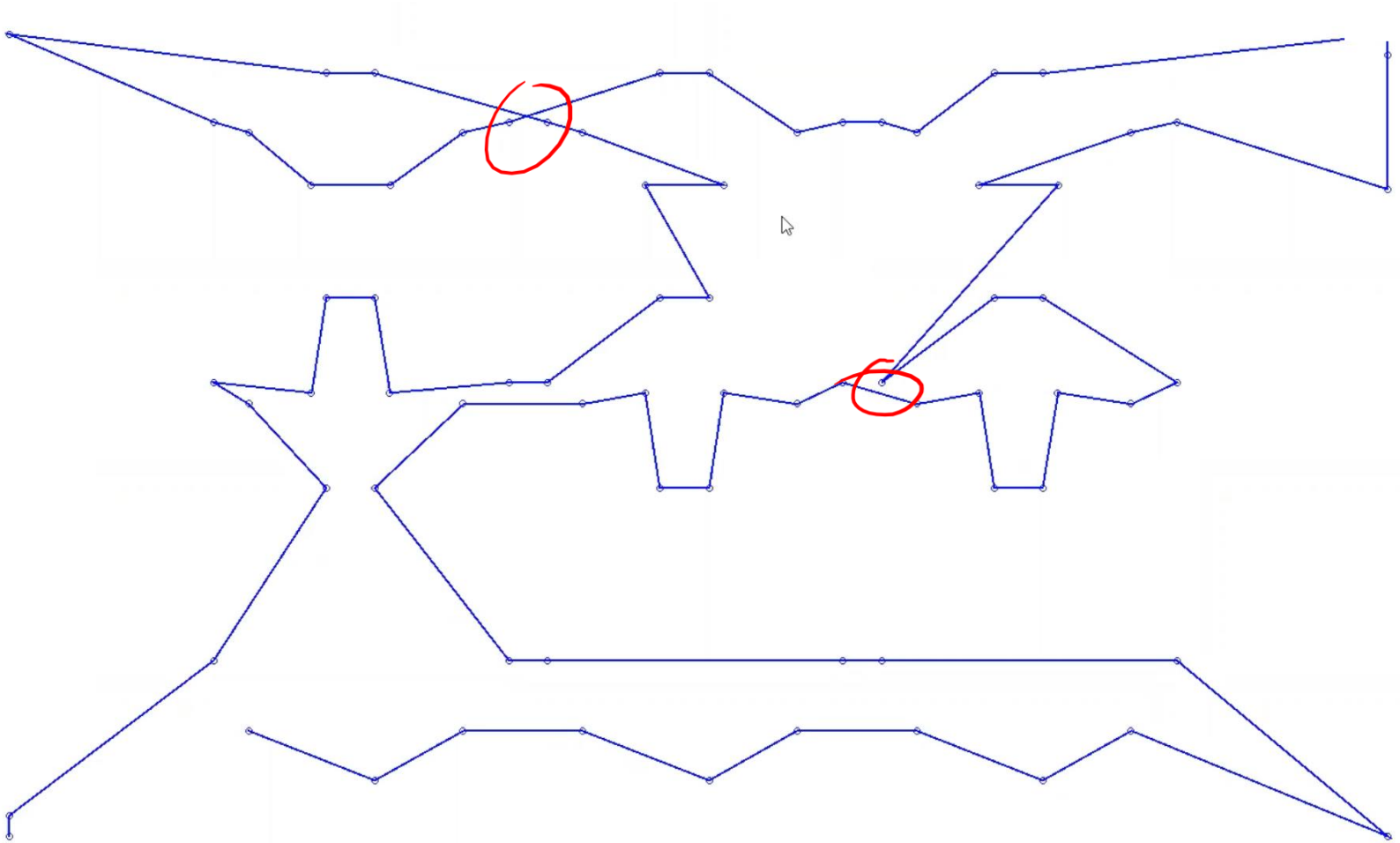
Construction algorithms – Minimum Insert Cost



Construction algorithms – Minimum Insert Cost



Construction algorithms – Minimum Insert Cost

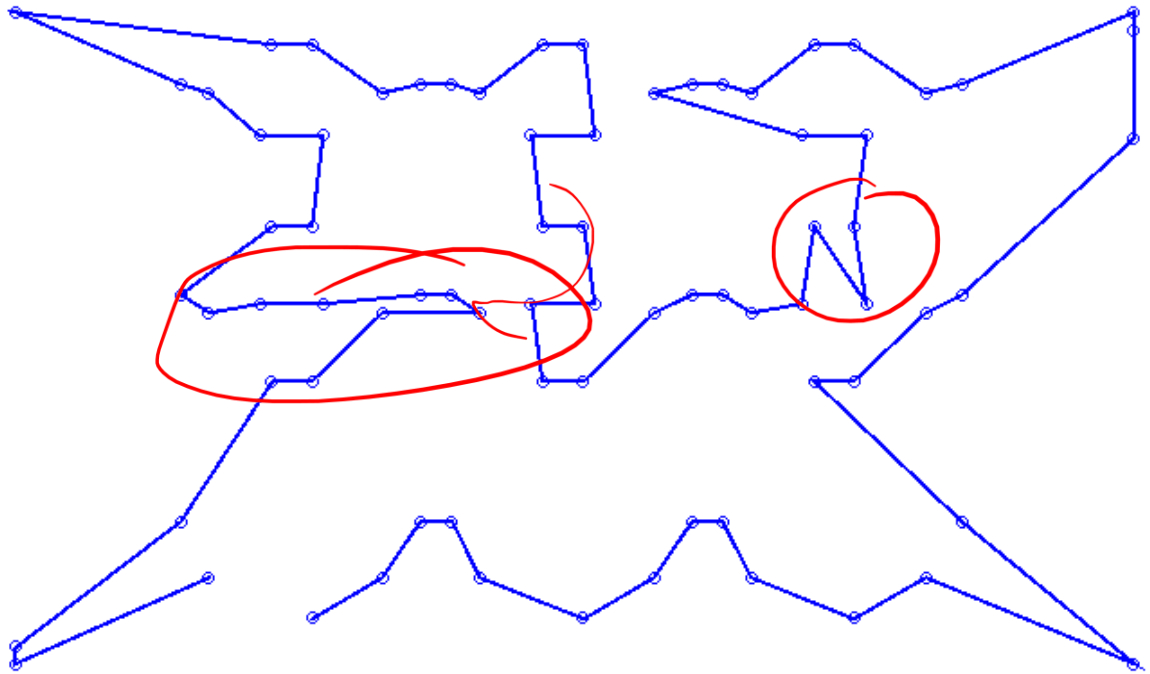


Construction algorithms – Randomisation

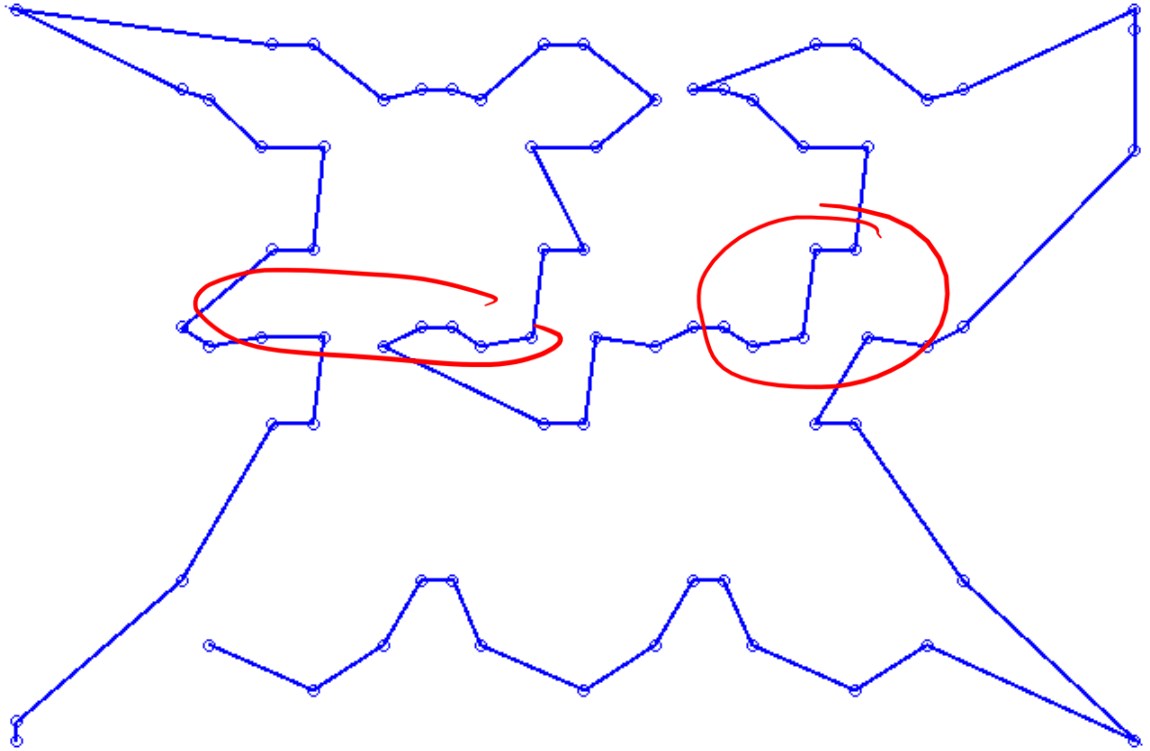
- At each iteration
  - Choose a **random customer**
  - **Insert it in the position that increases the cost by the least**
  - Allows insert between *any* pair of customers – not just at the end

Construction algorithms–Randomisation

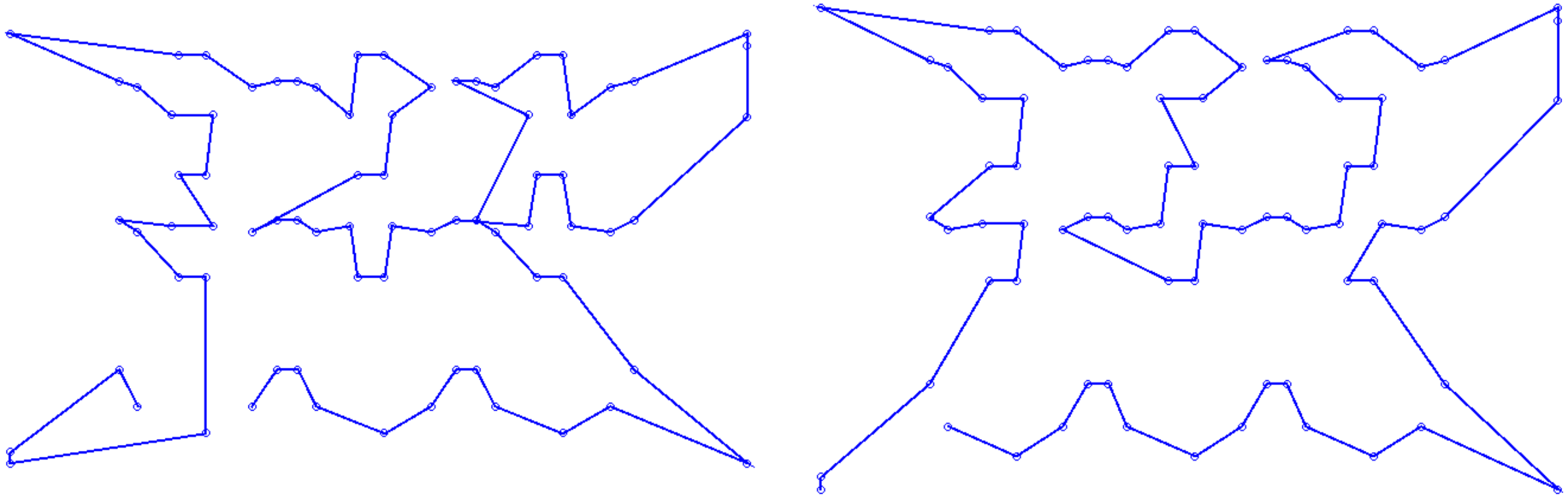
ONE RANDOM TOUR



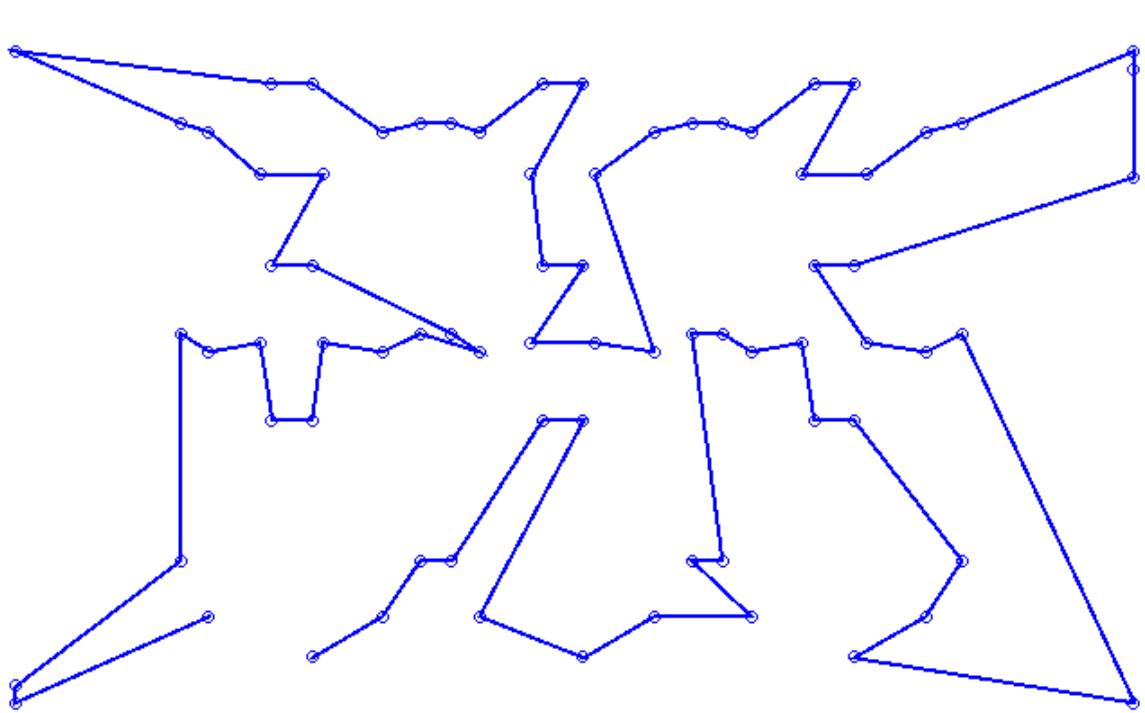
OPT. TOUR

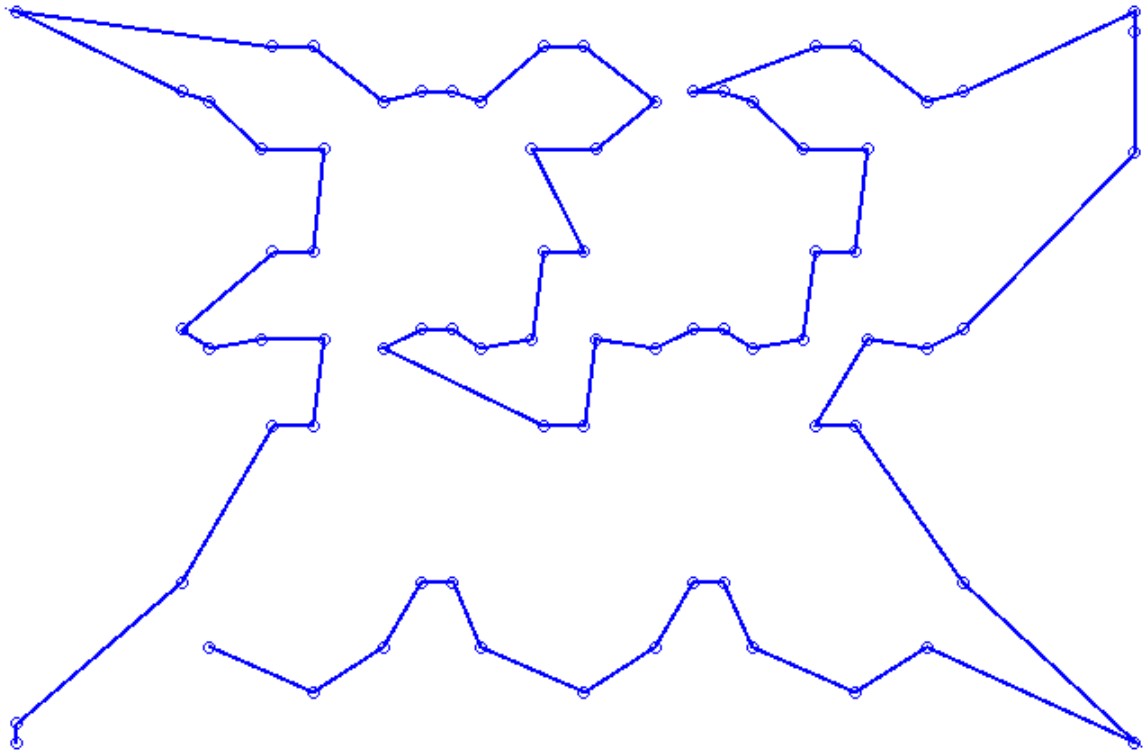


Construction algorithms–Randomisation

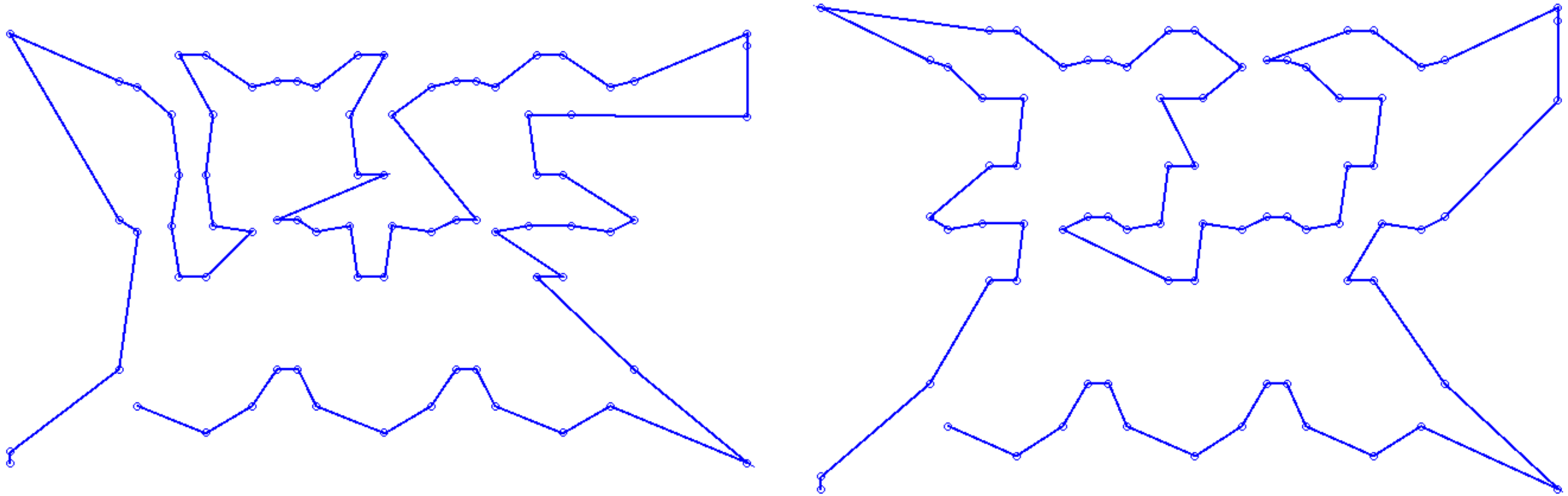


Construction algorithms–Randomisation

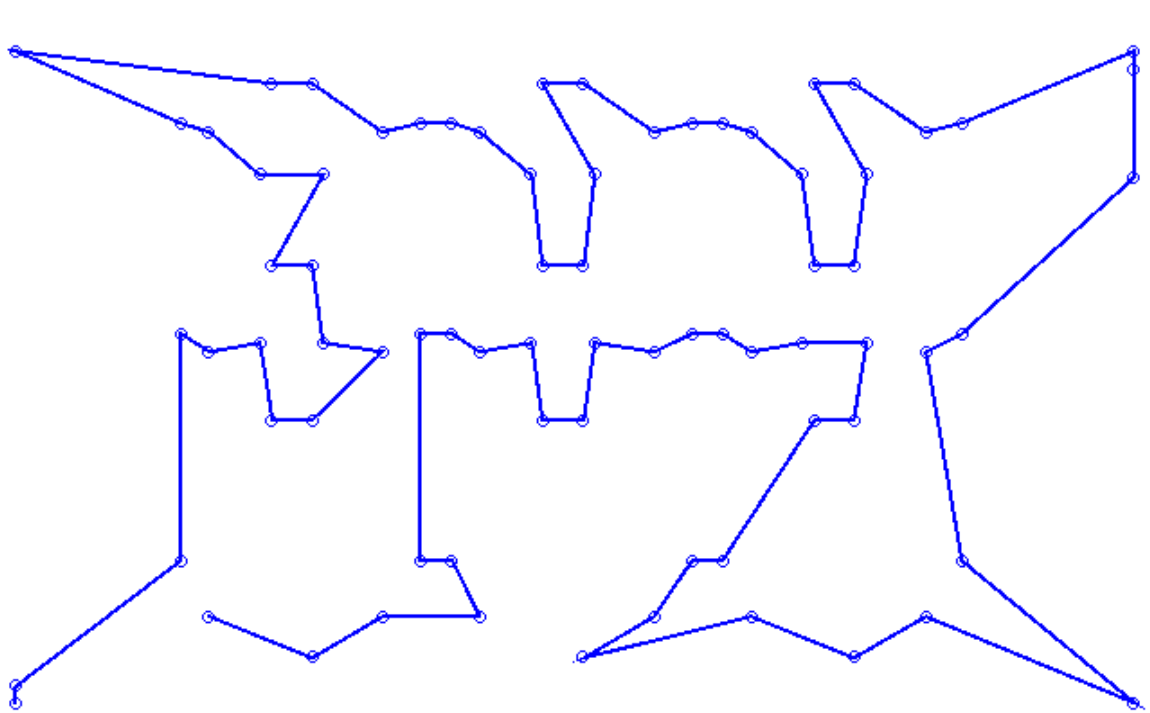


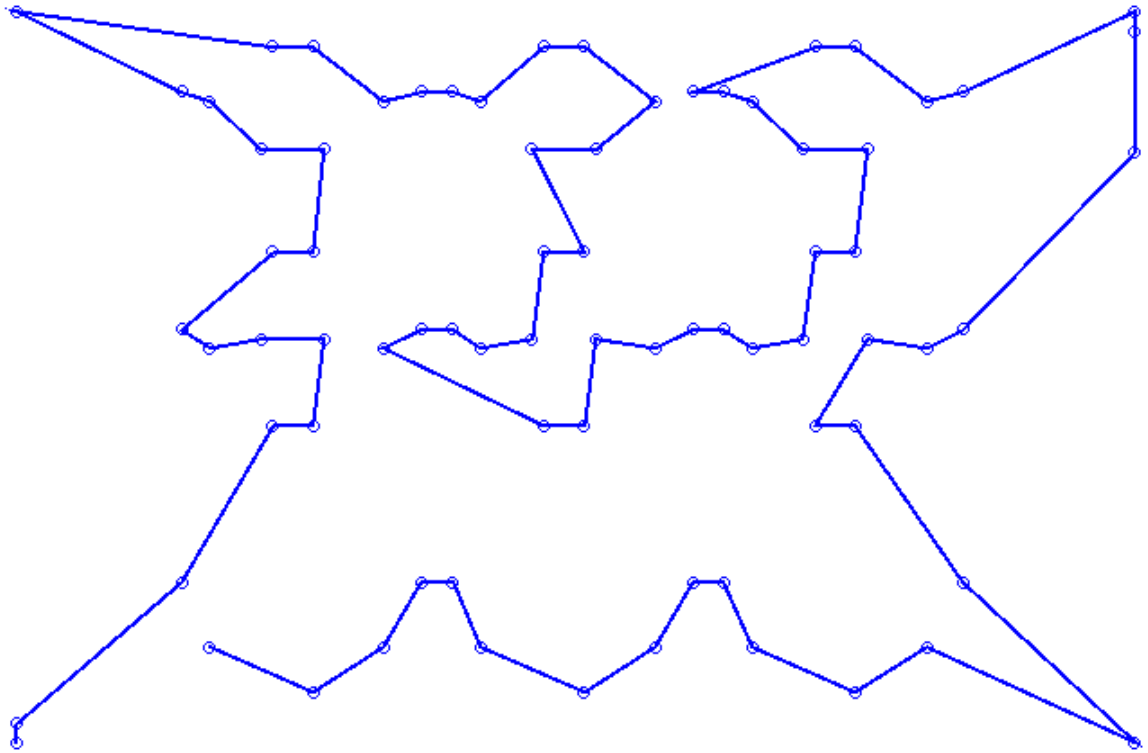


Construction algorithms–Randomisation

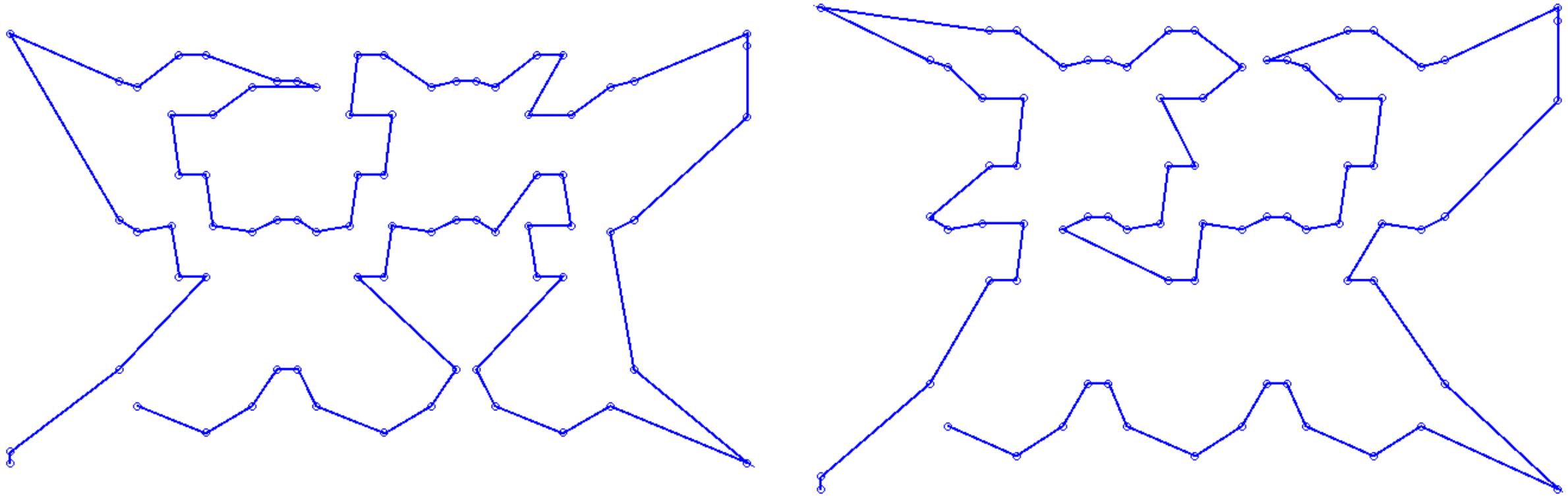


Construction algorithms–Randomisation

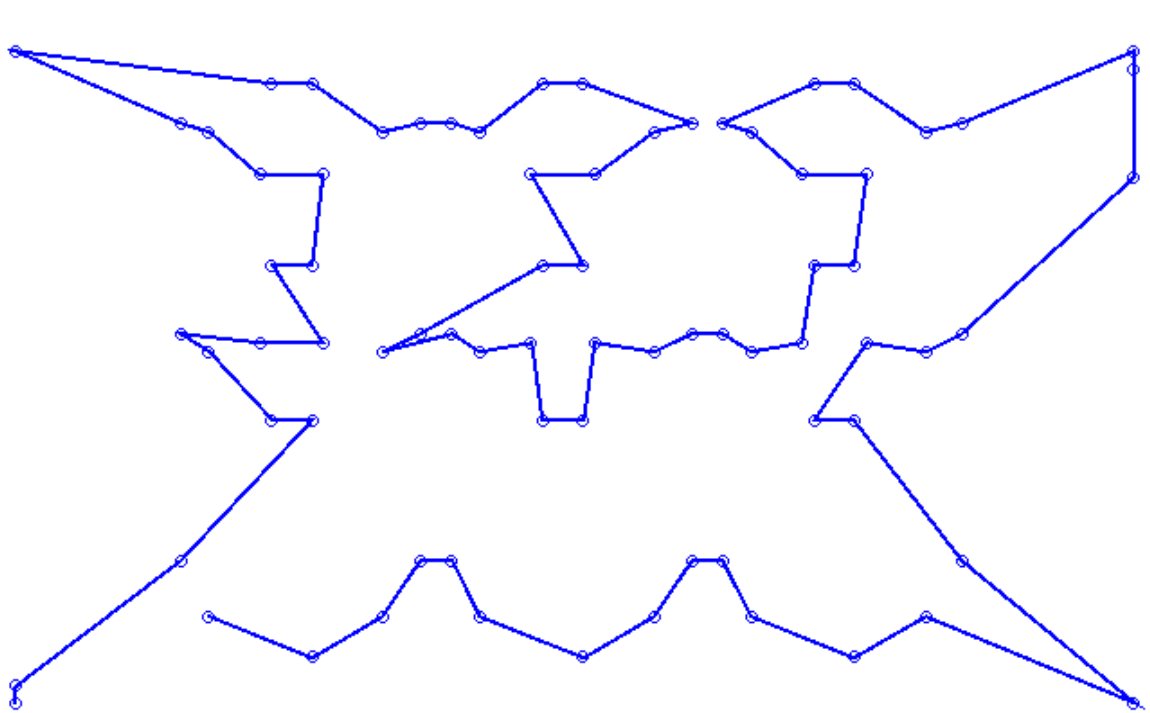


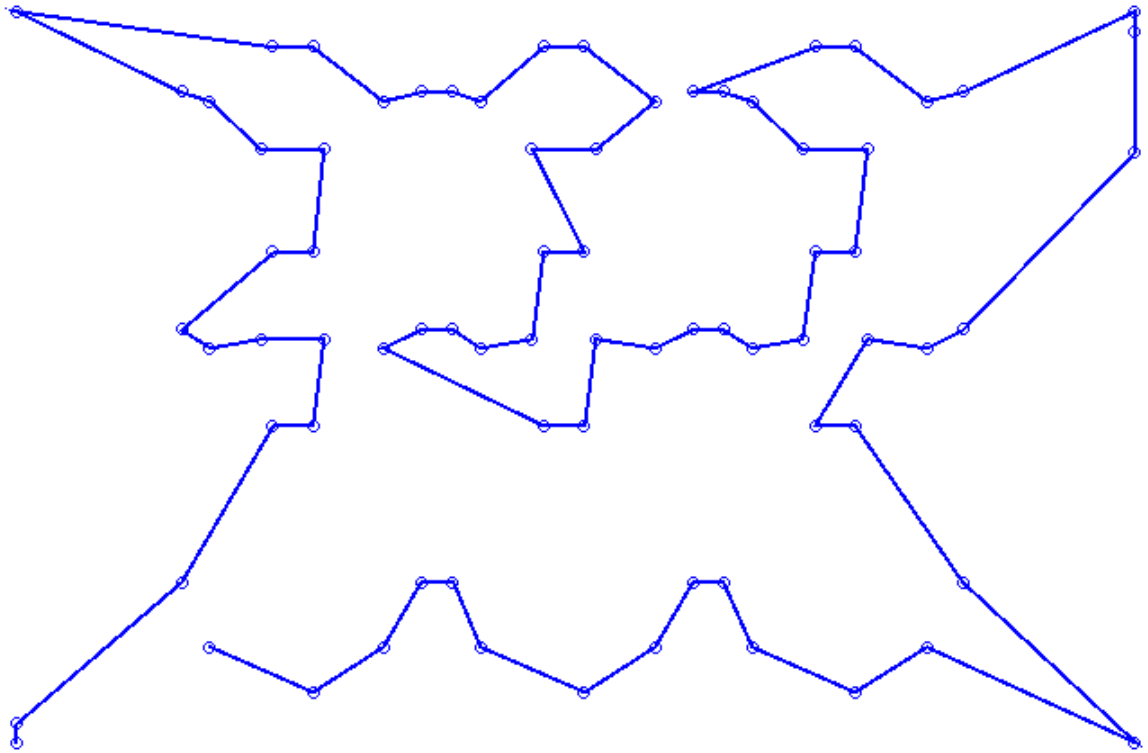


Construction algorithms–Randomisation



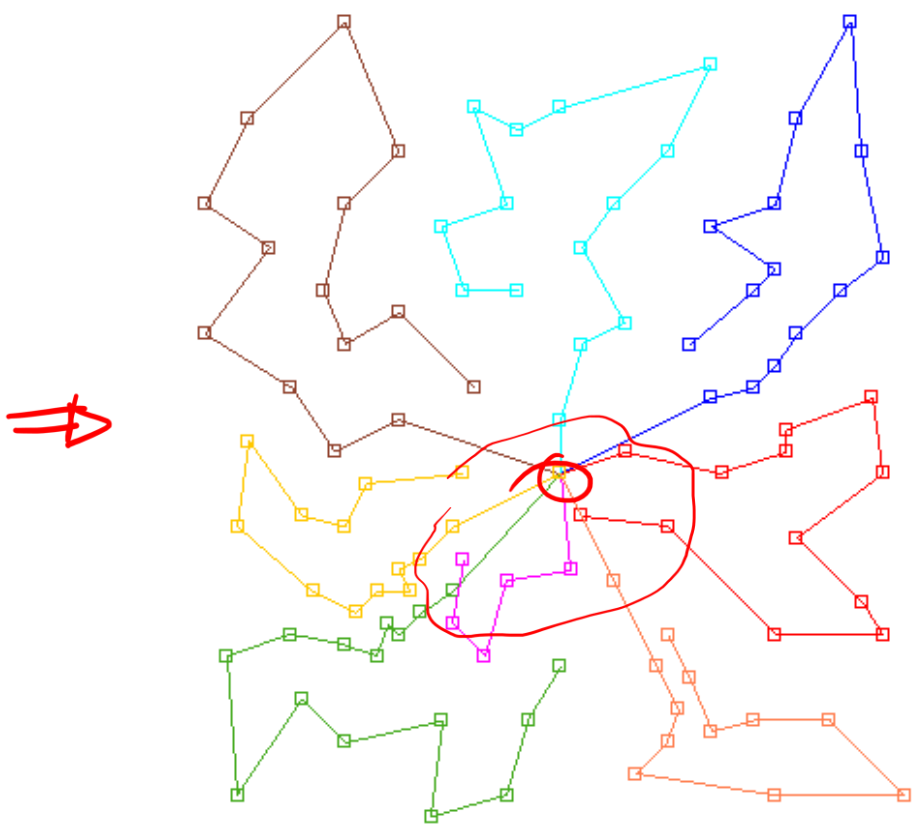
Construction algorithms–Randomisation

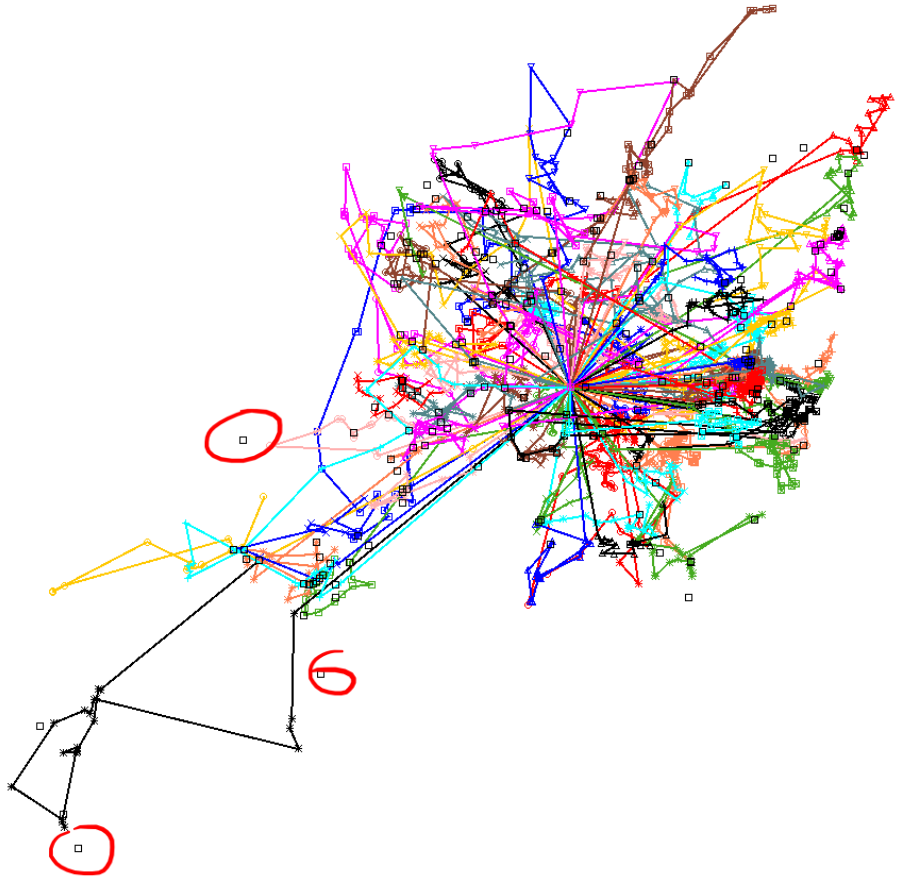




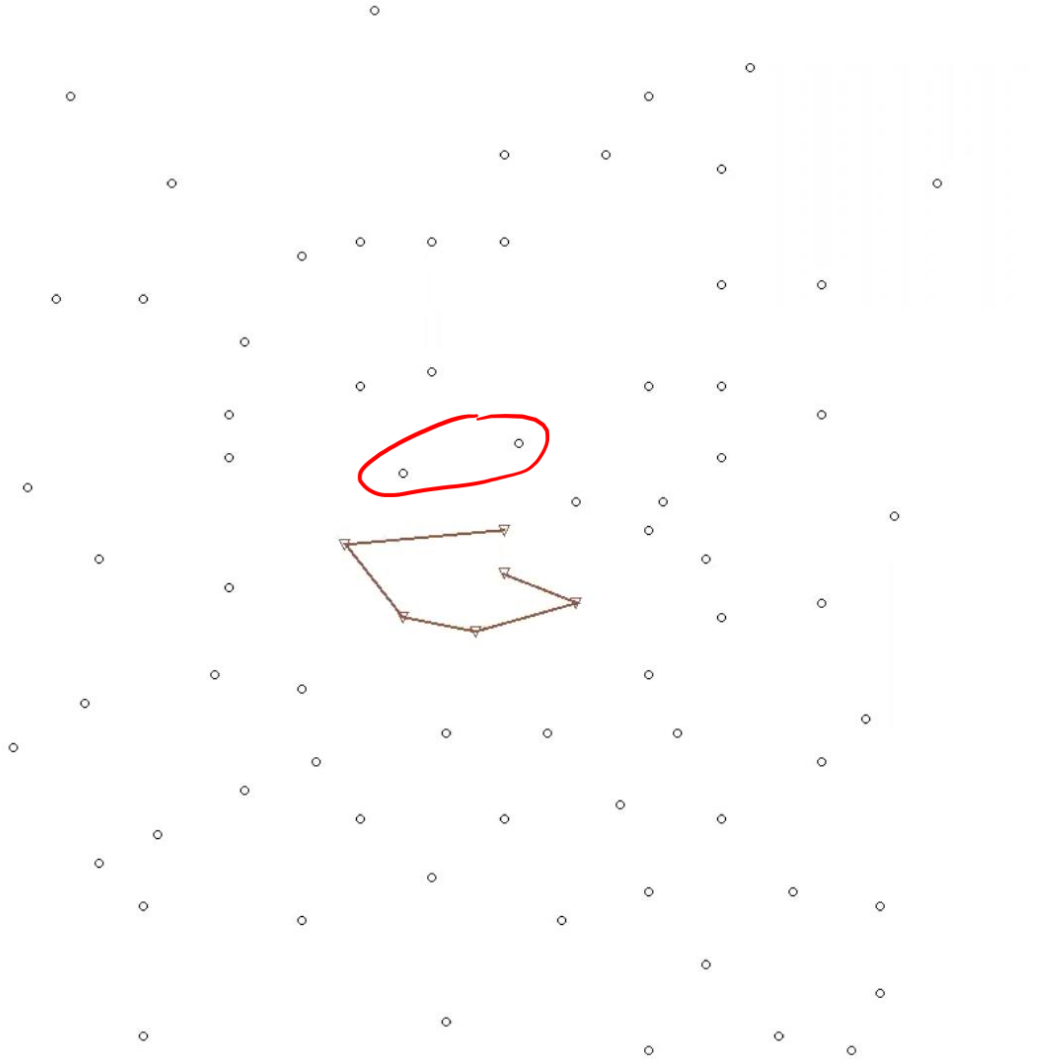
Vehicle Routing Problem (VRP)

Given a set of customers, and a fleet of vehicles to make deliveries, find a set of routes that services all customers at minimum cost

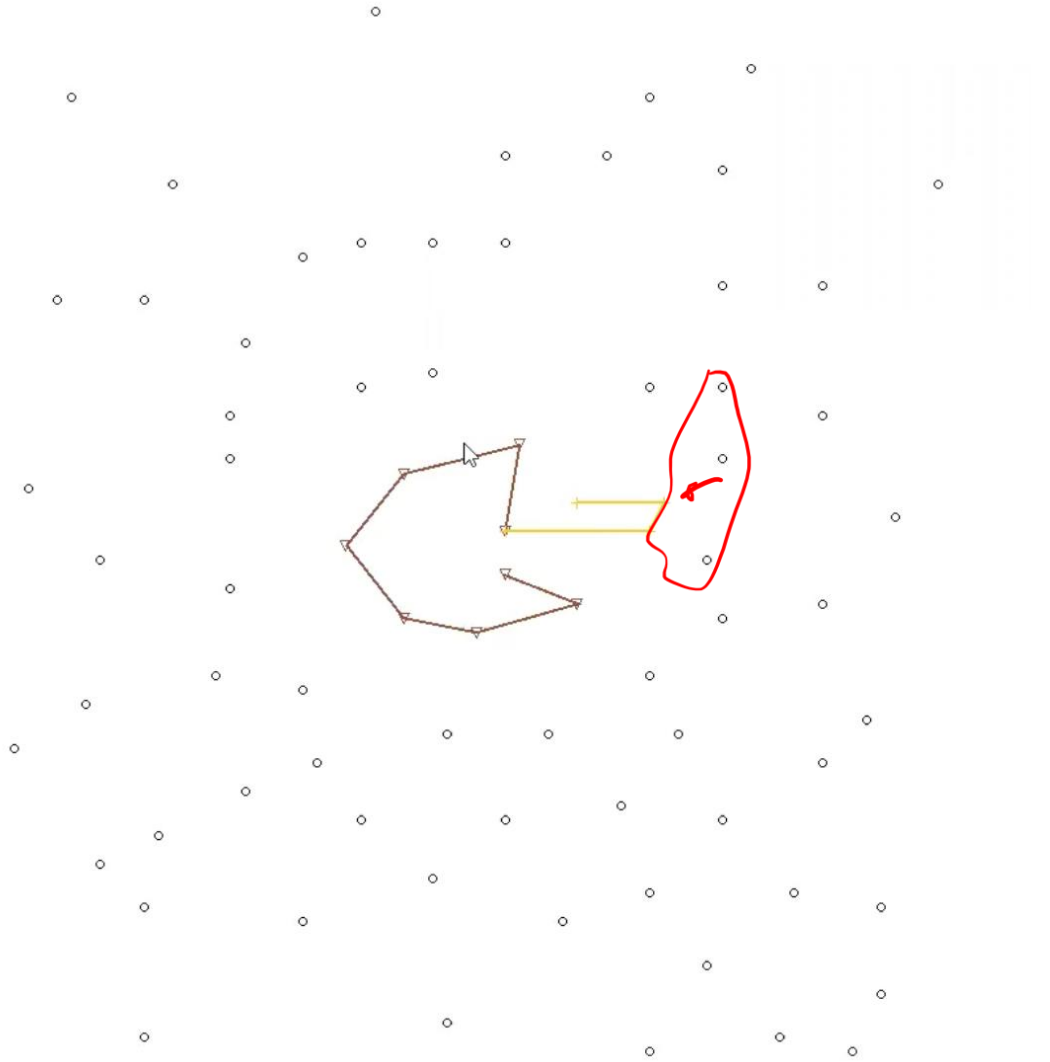




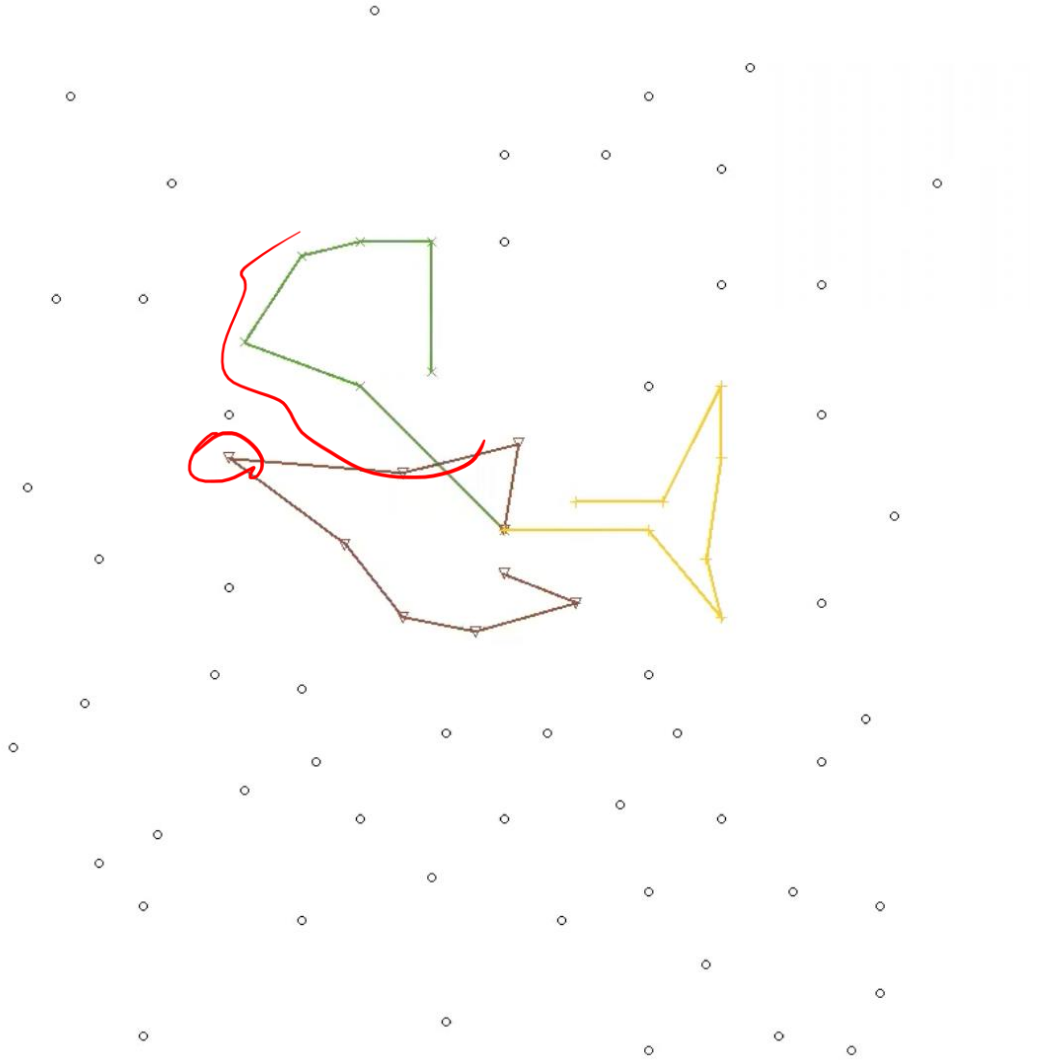
Minimum Insert Cost – VRP



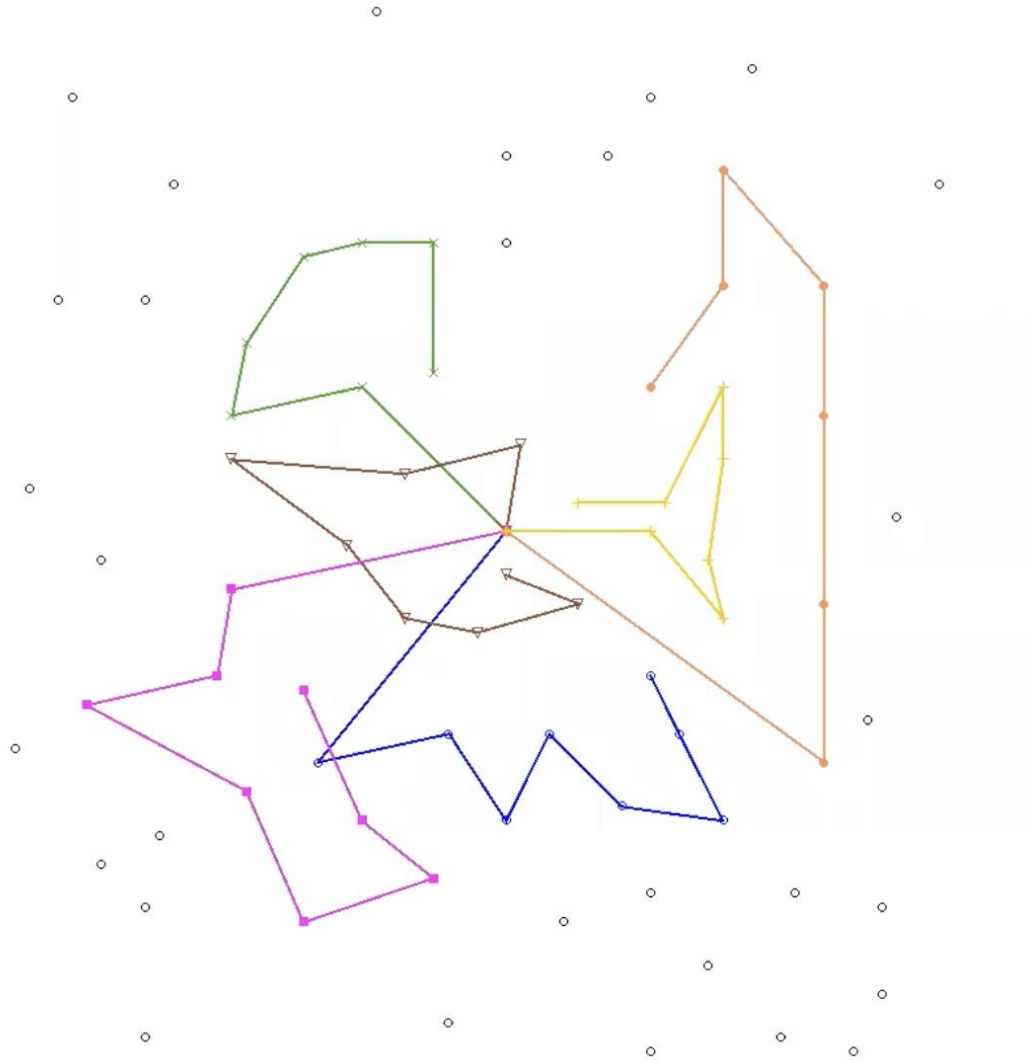
Minimum Insert Cost – VRP



Minimum Insert Cost – VRP

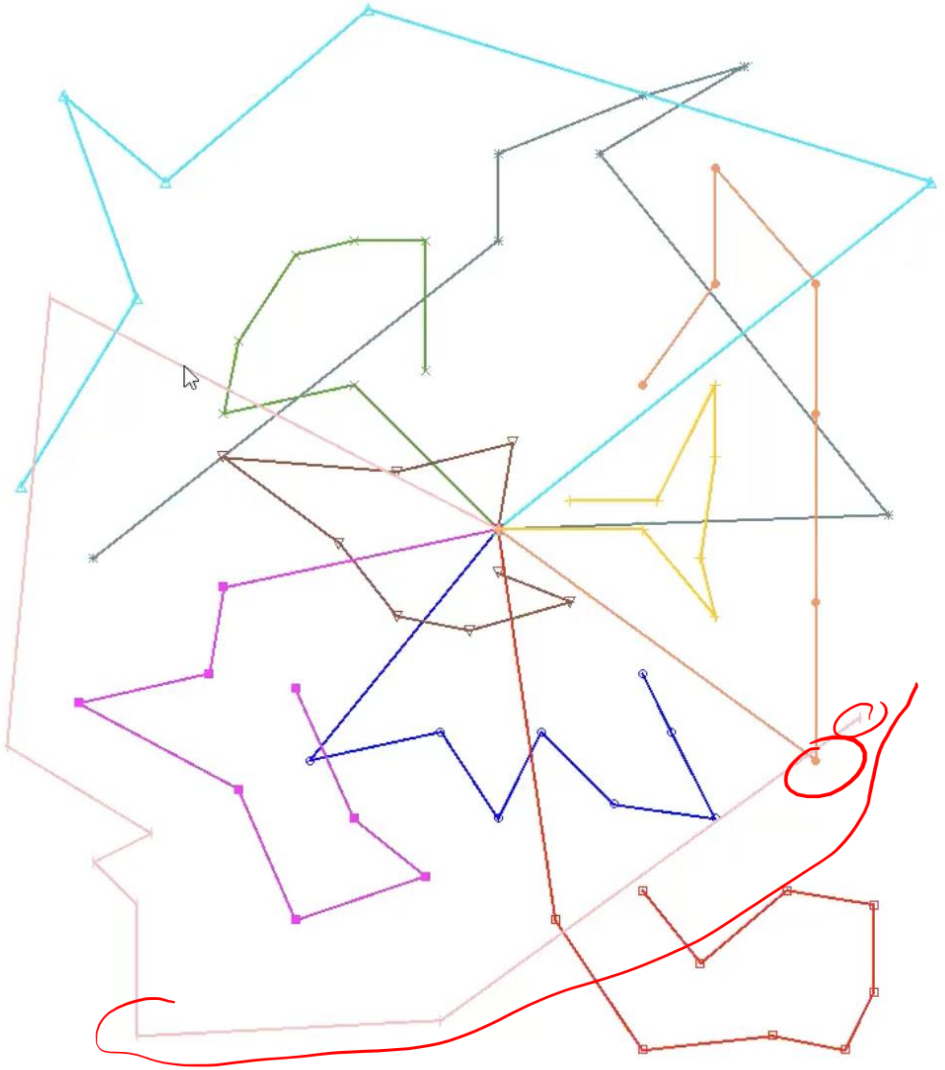


Minimum Insert Cost – VRP



Minimum Insert Cost – VRP

COMBINING
H/MH + MATH OPT
AI: LNS ← NGT WGGK
LARGE \ N. \ SEARCH

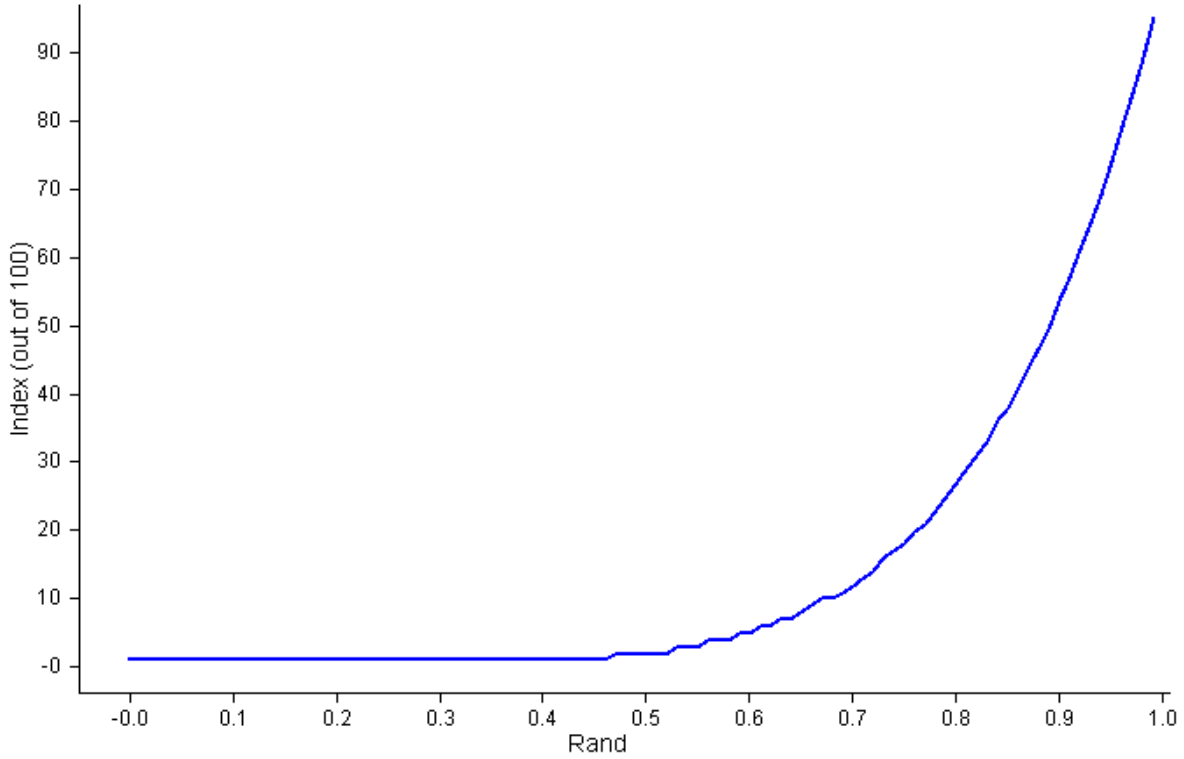


randomising the order does not help ☹

Construction algorithms

- Combine “greedy” and “random”
- Don’t always make the very “greediest” choice

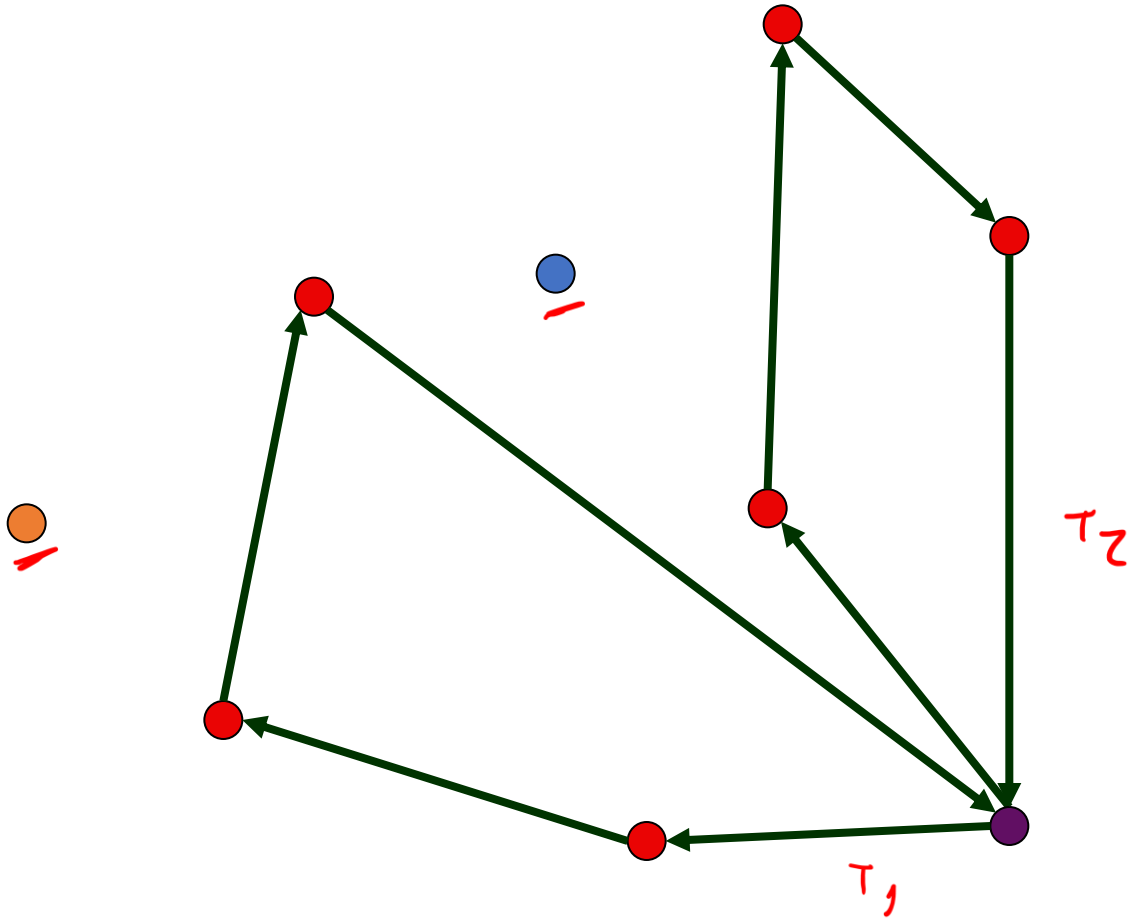
- E.g. choose the r^{th} best of n choices
- $r = n * (\text{uniform}(0,1))^{y}$



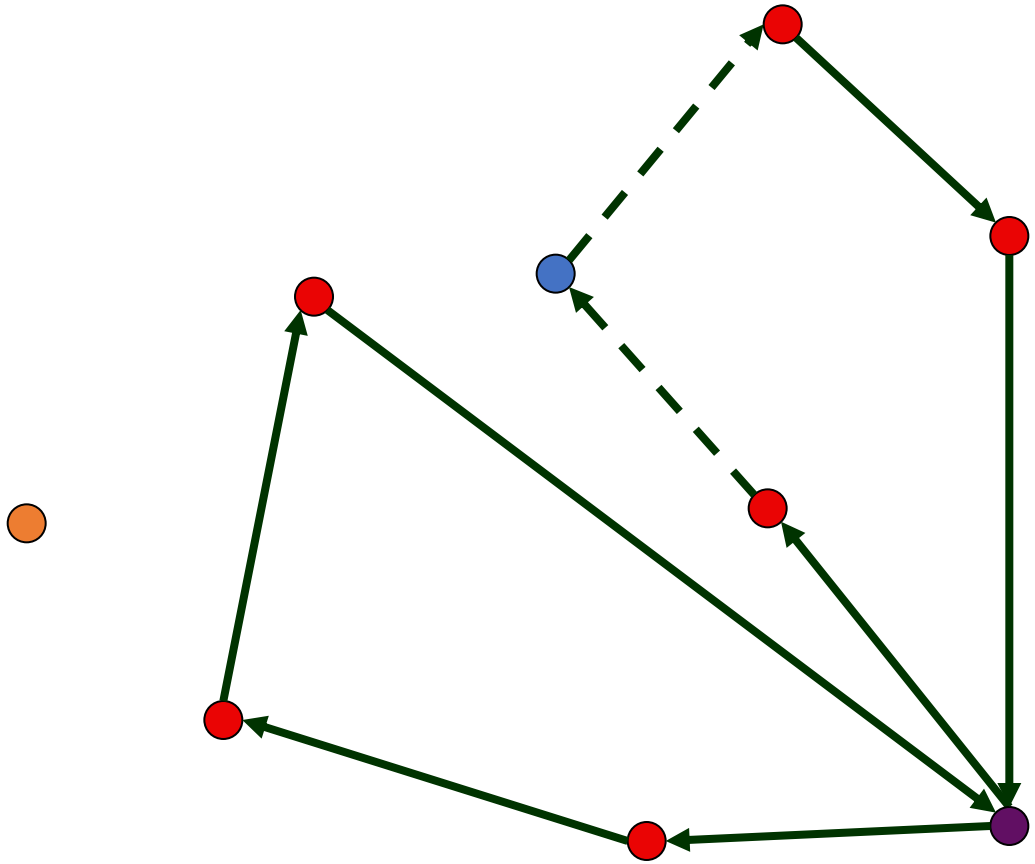
Construction algorithms

- We have seen that greedy algorithms lack “foresight”
- Next step: Regret

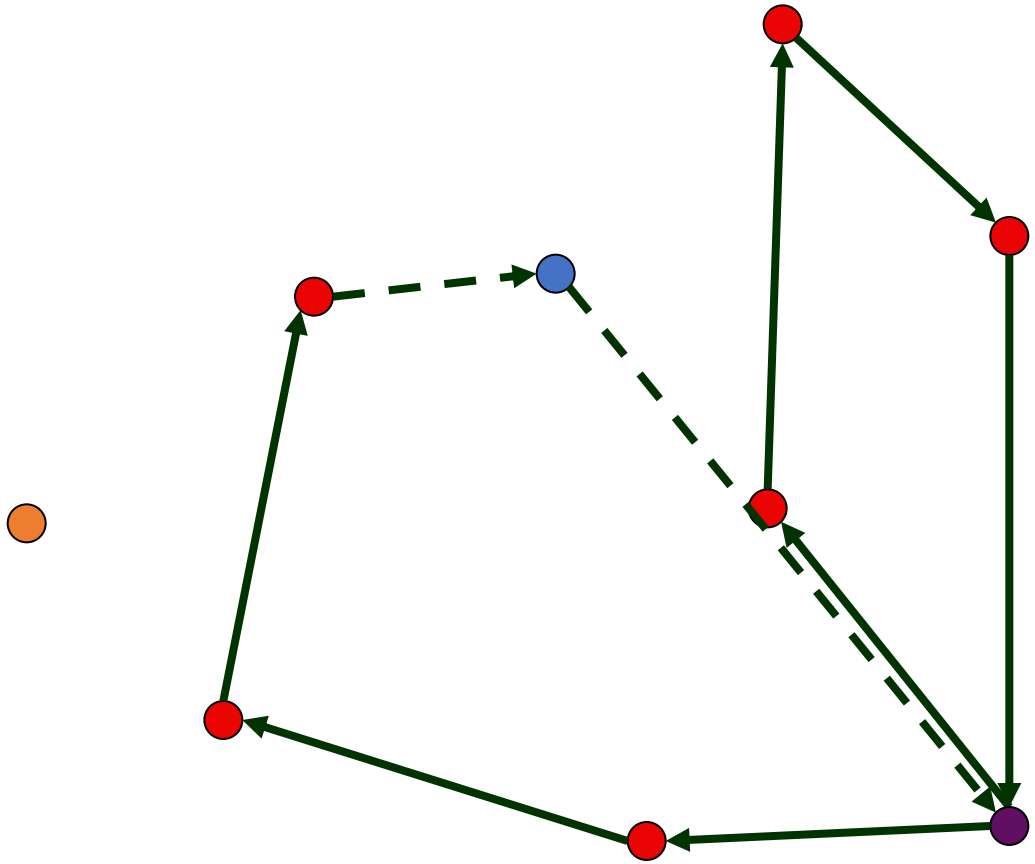
Regret – VRP



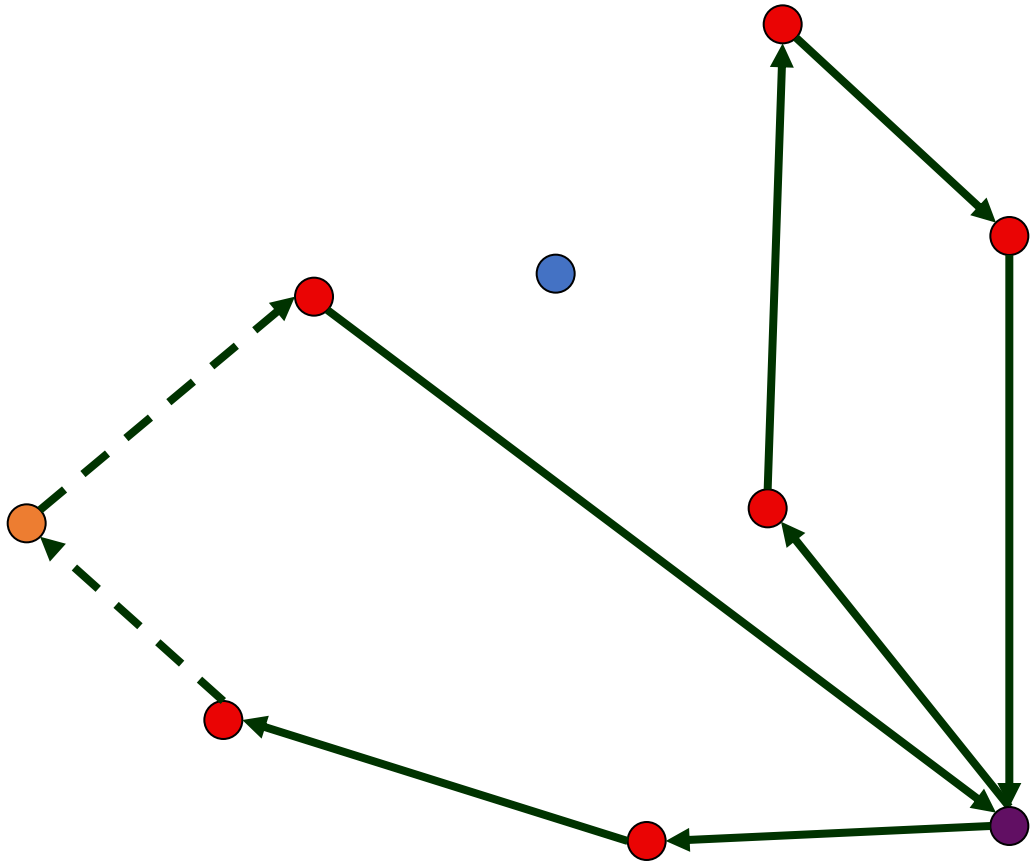
Regret – VRP



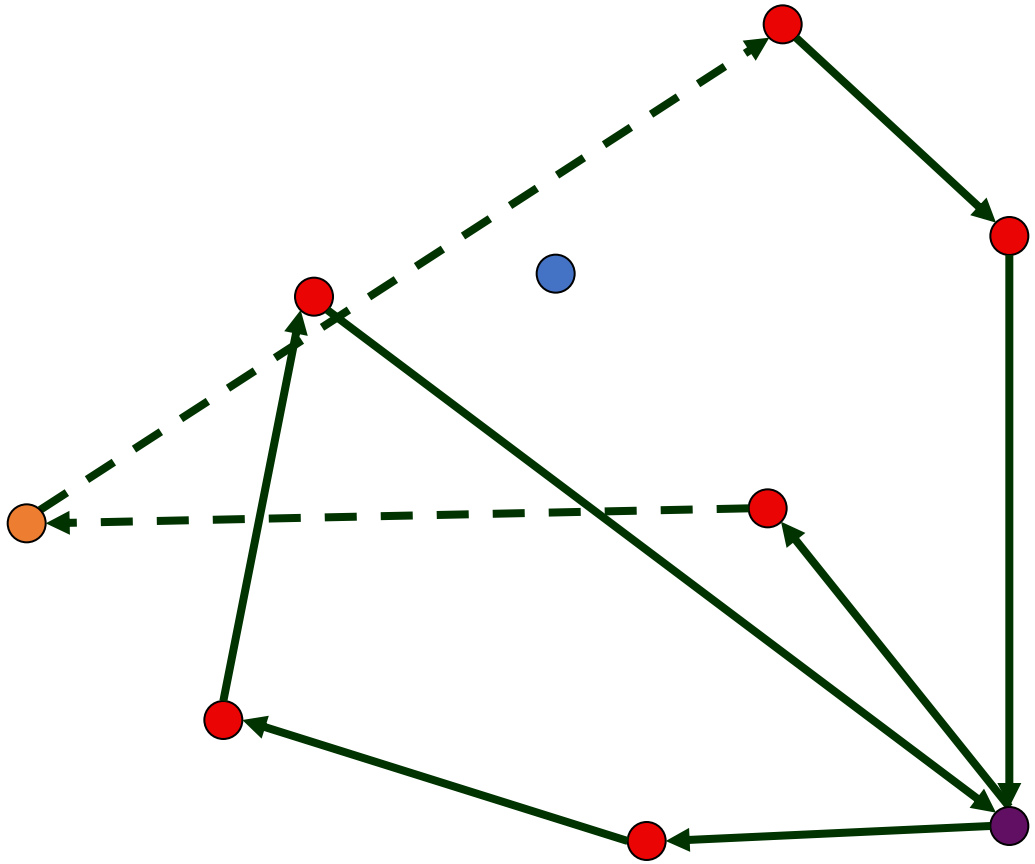
Regret – VRP



Regret – VRP



Regret – VRP



Regret

regret(i) = C(insert i in 2nd-best route) – C(insert i in best route)

= $f(2,i) - f(1,i)$

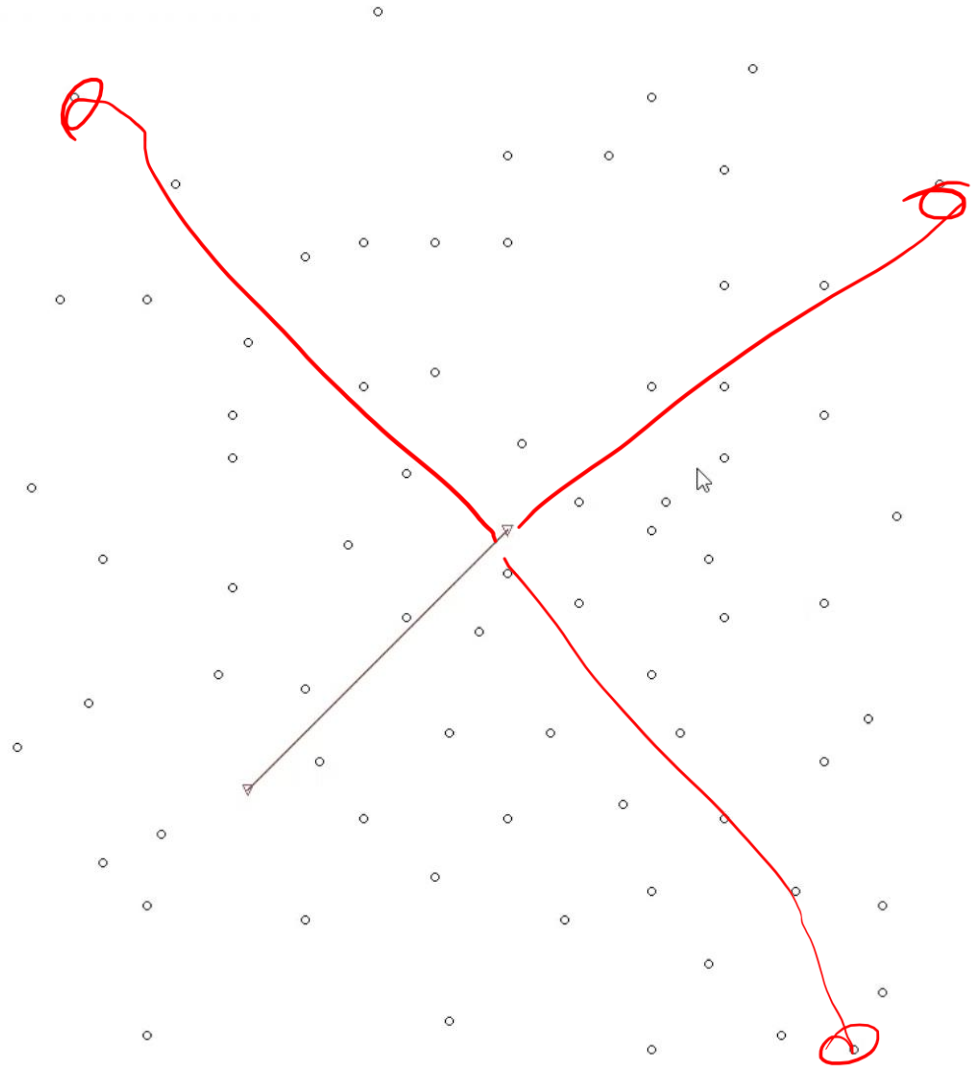
Select the choice i with largest regret

- VRP: next customer to be served is the one with largest regret

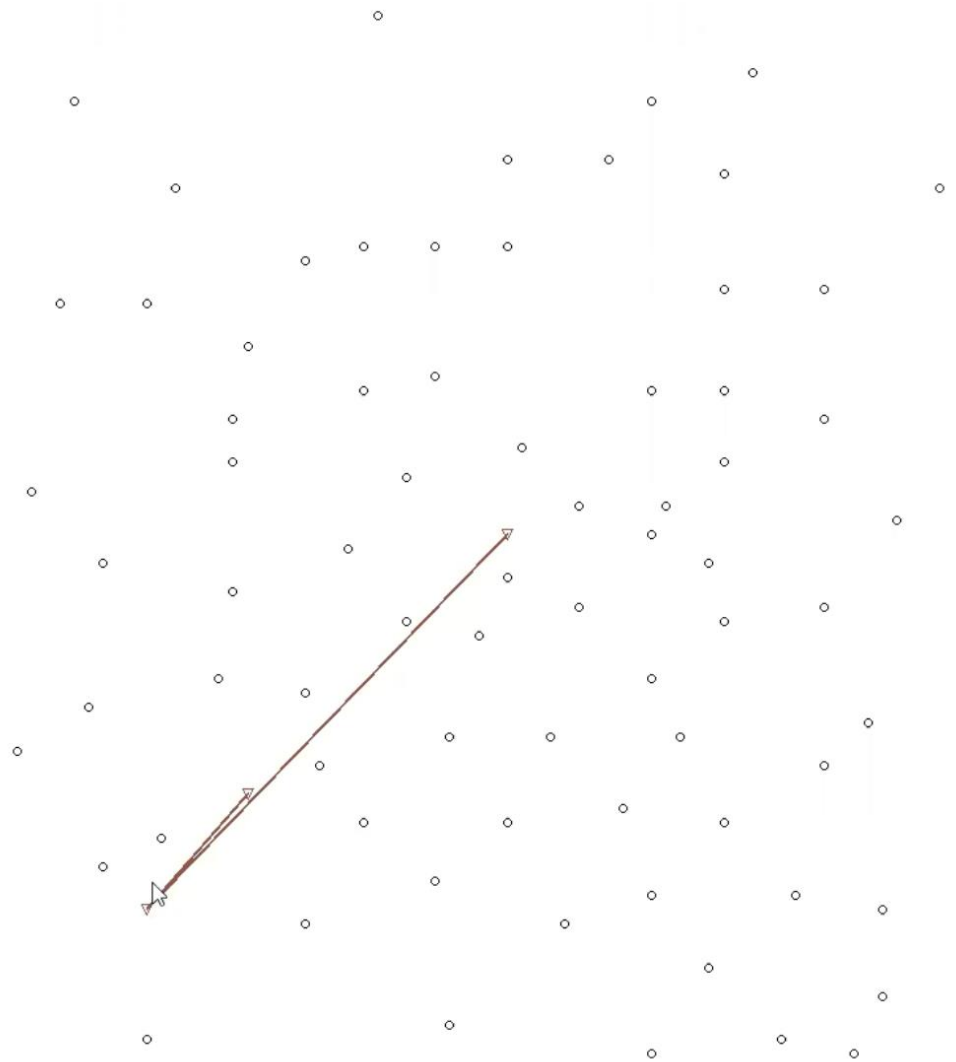
Can be extended to the k-th best routes:

k-regret (i) = $\sum_{j=2}^{k} (f(j,i) - f(1,i))$

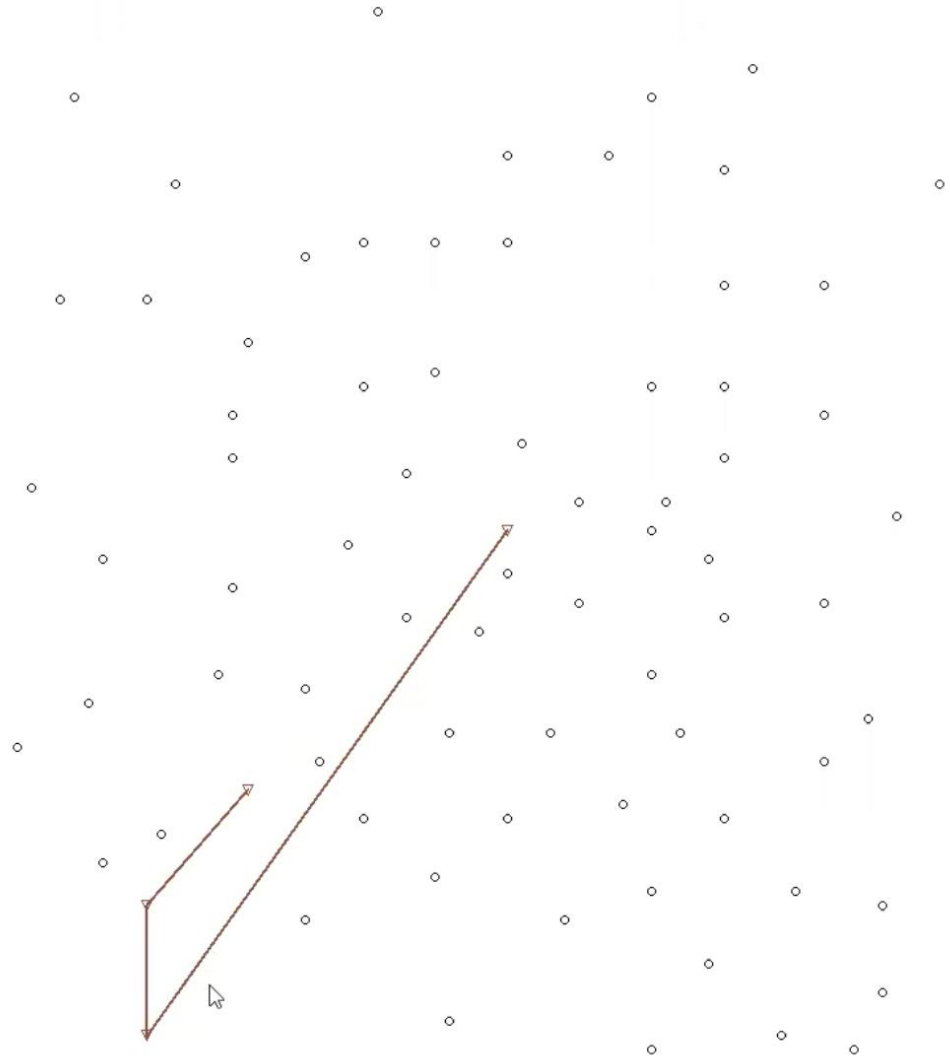
Regret



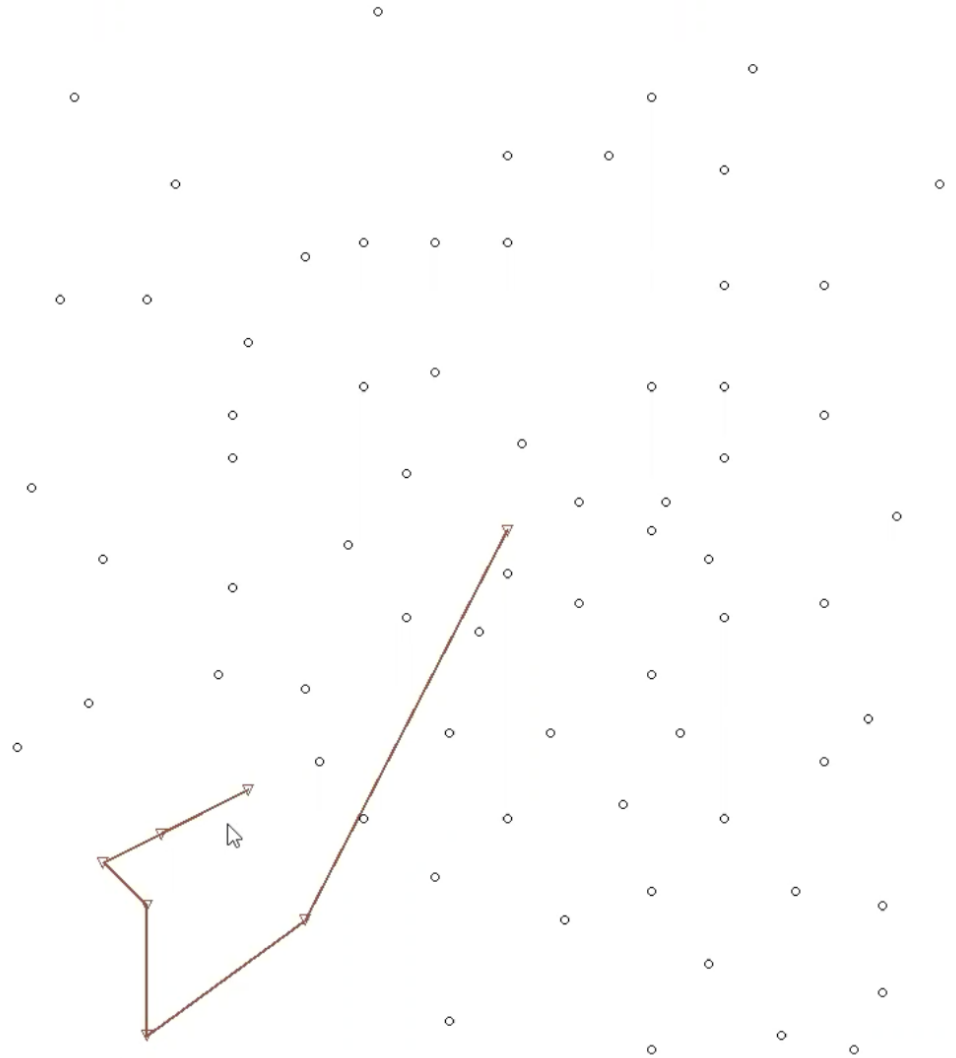
Regret



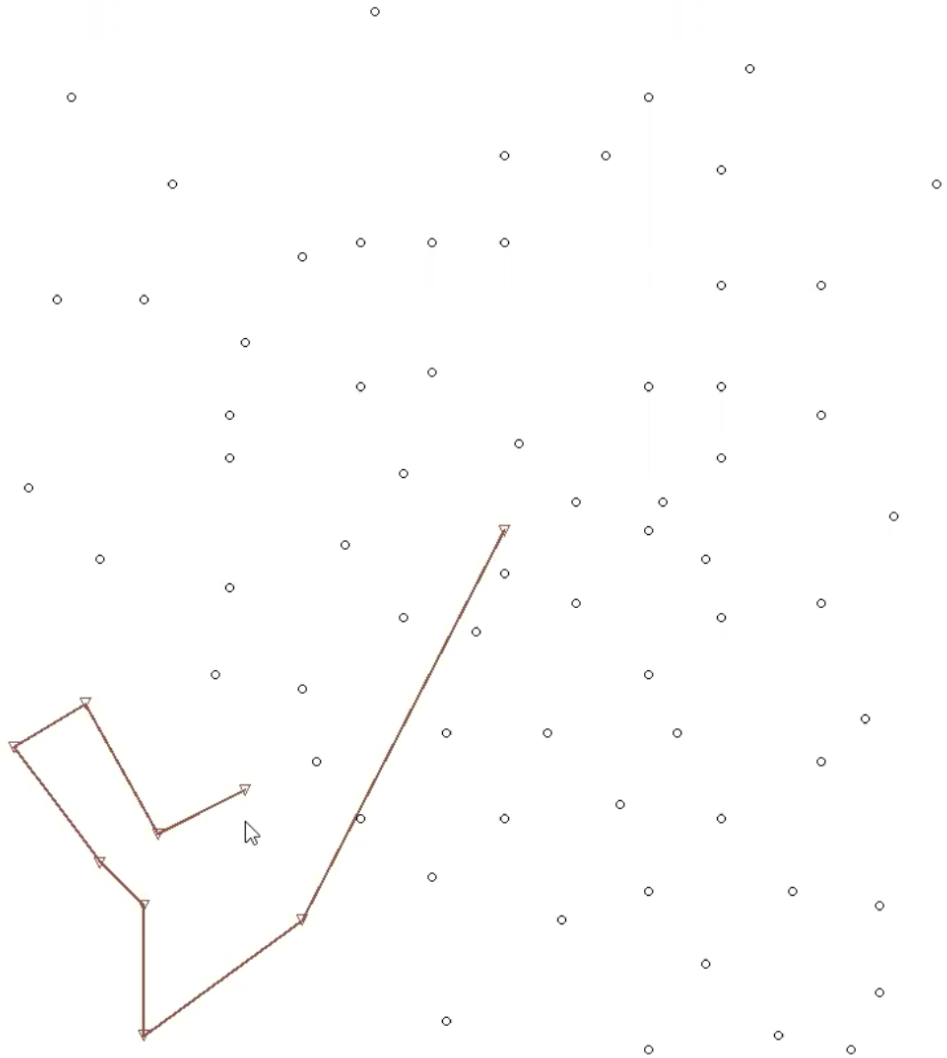
Regret



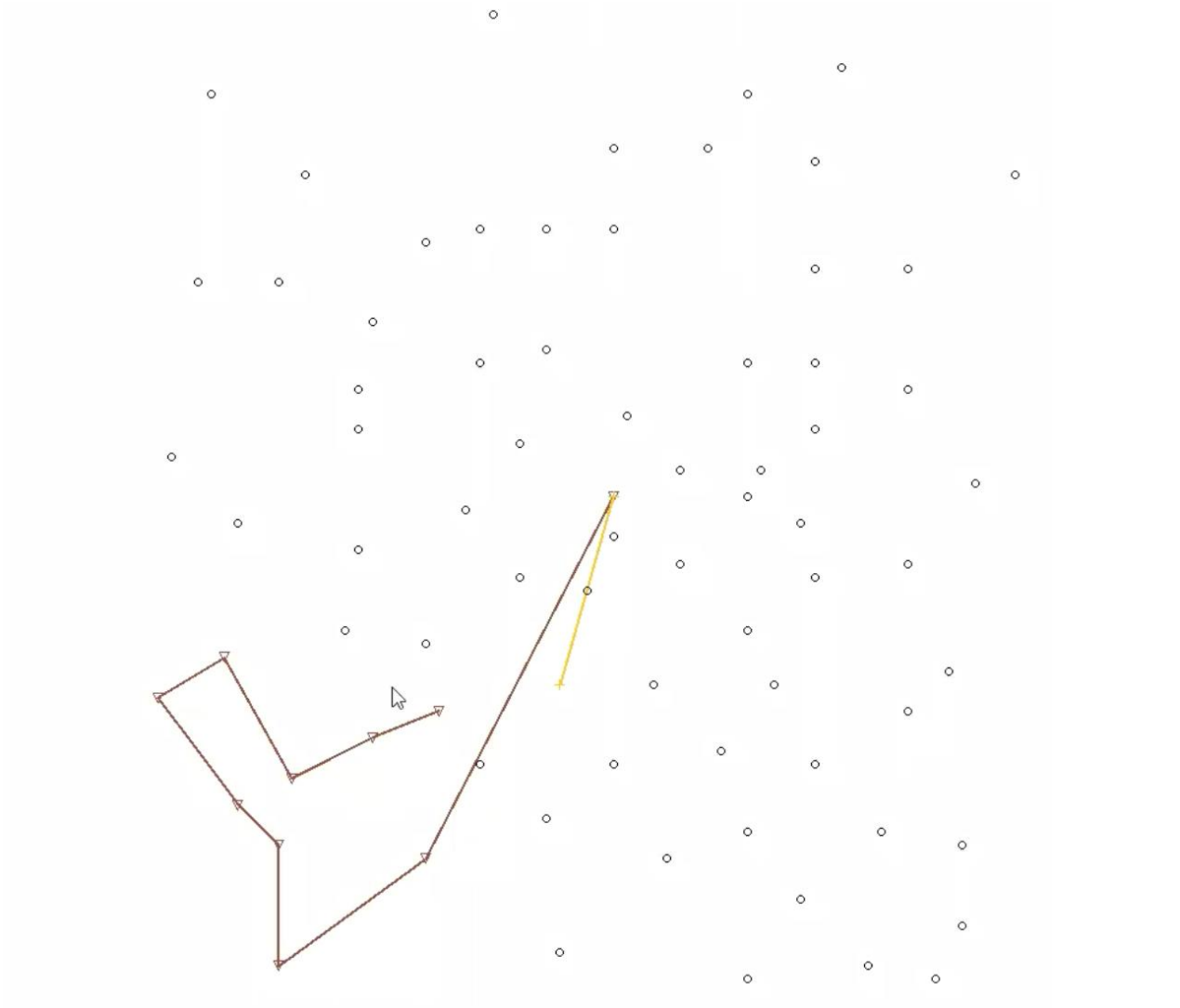
Regret



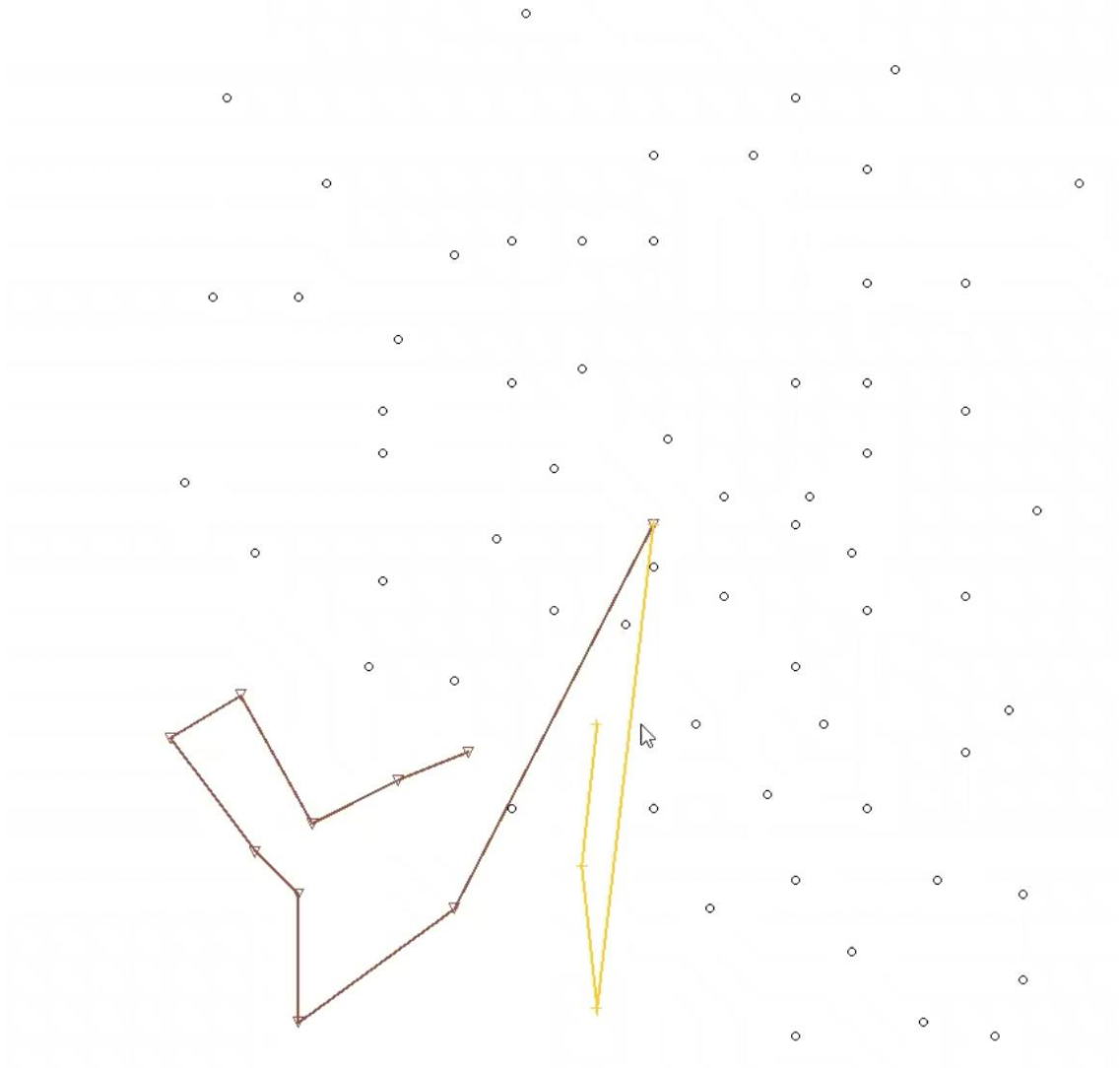
Regret



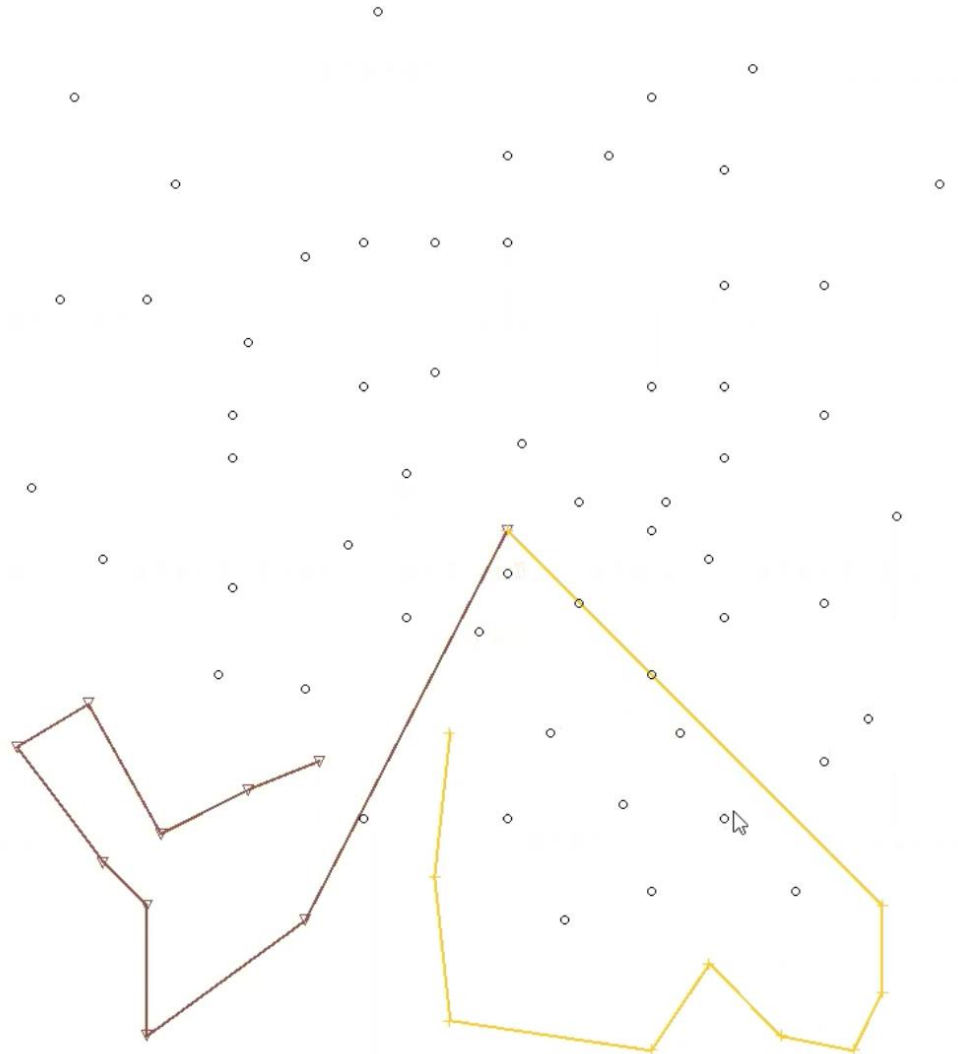
Regret



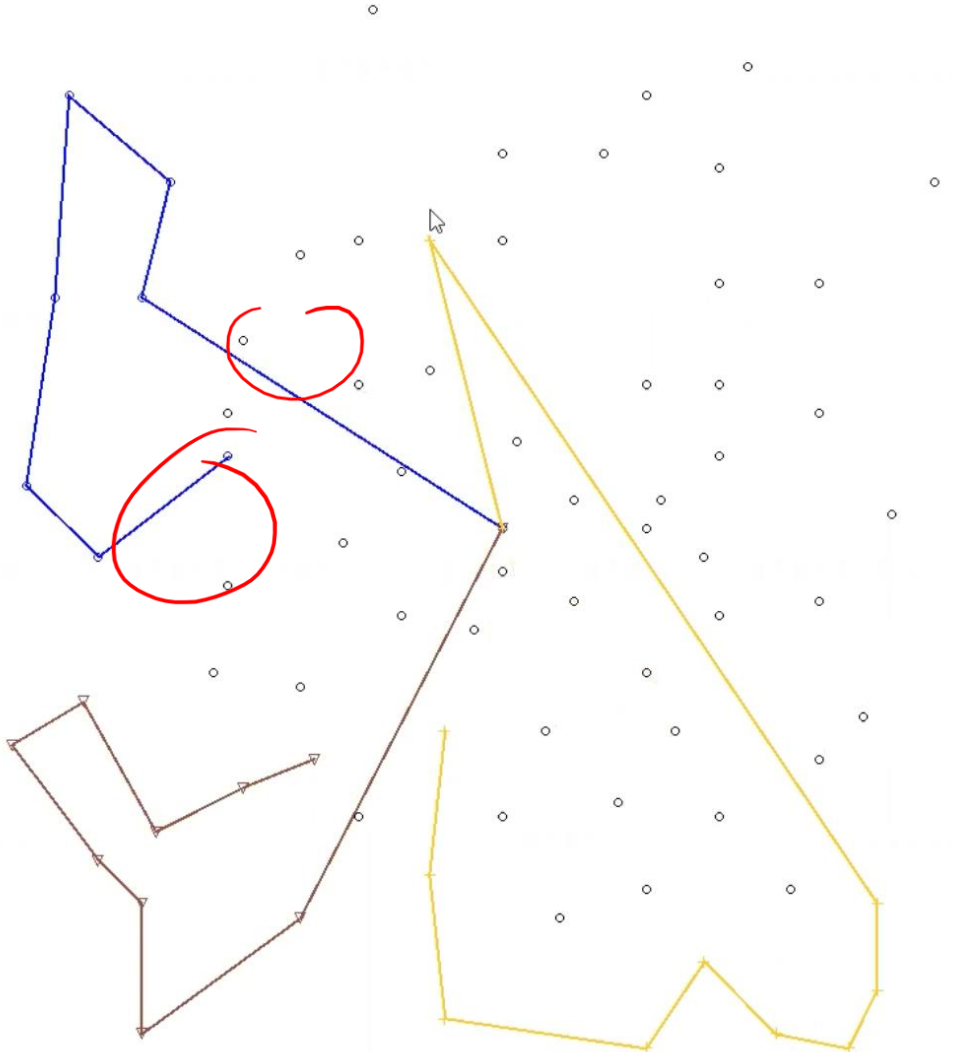
Regret



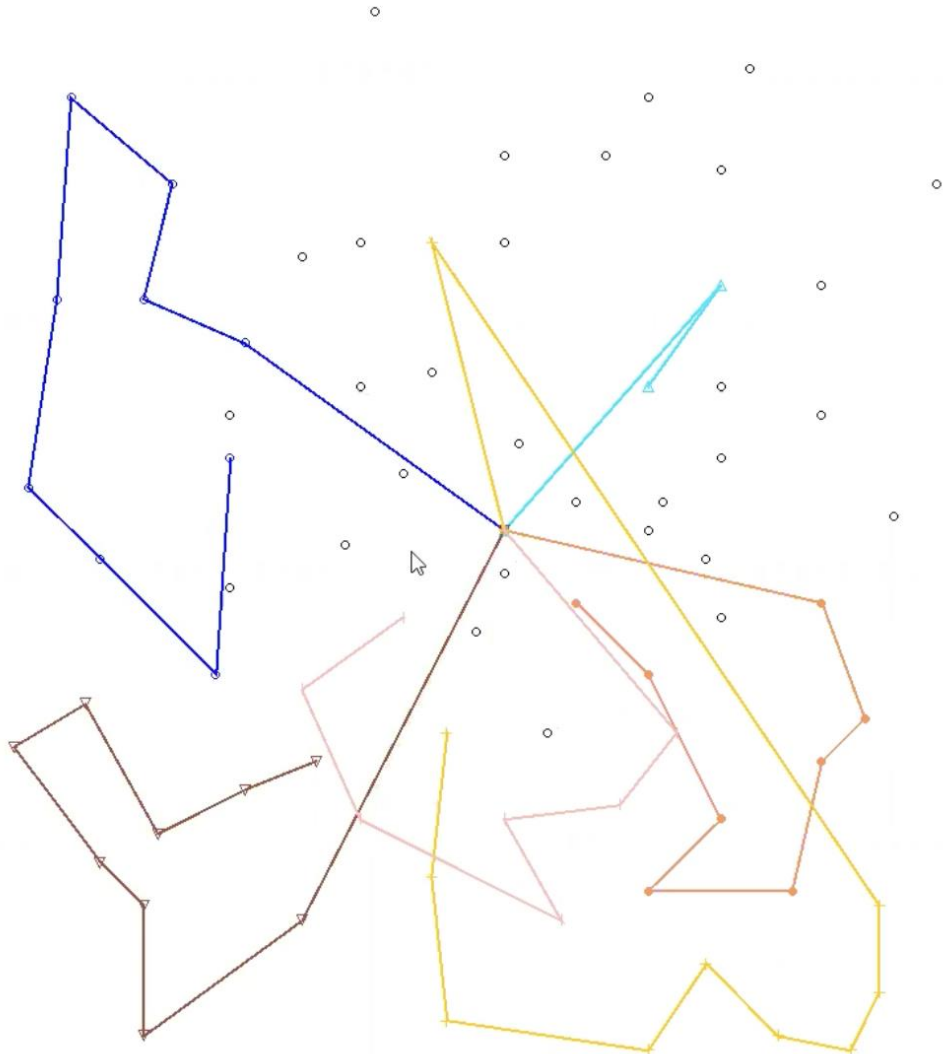
Regret



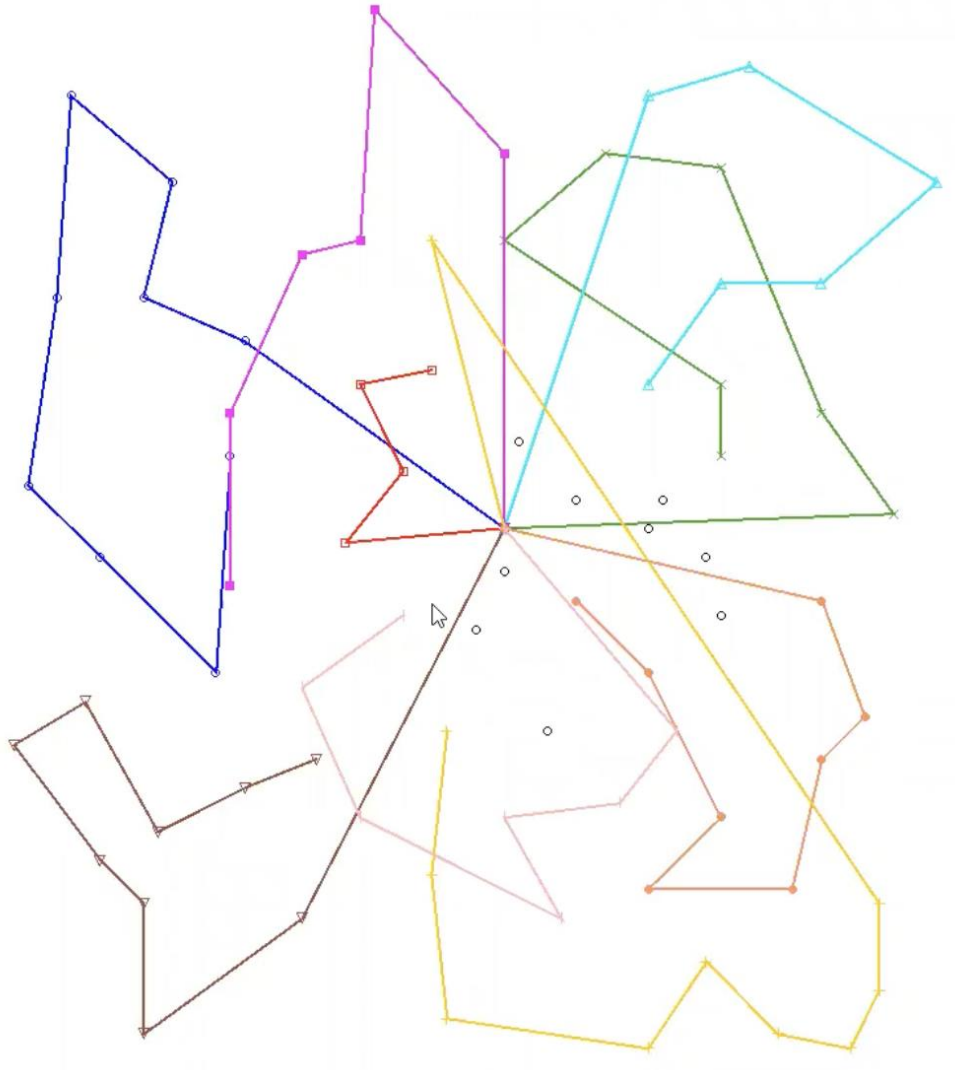
Regret



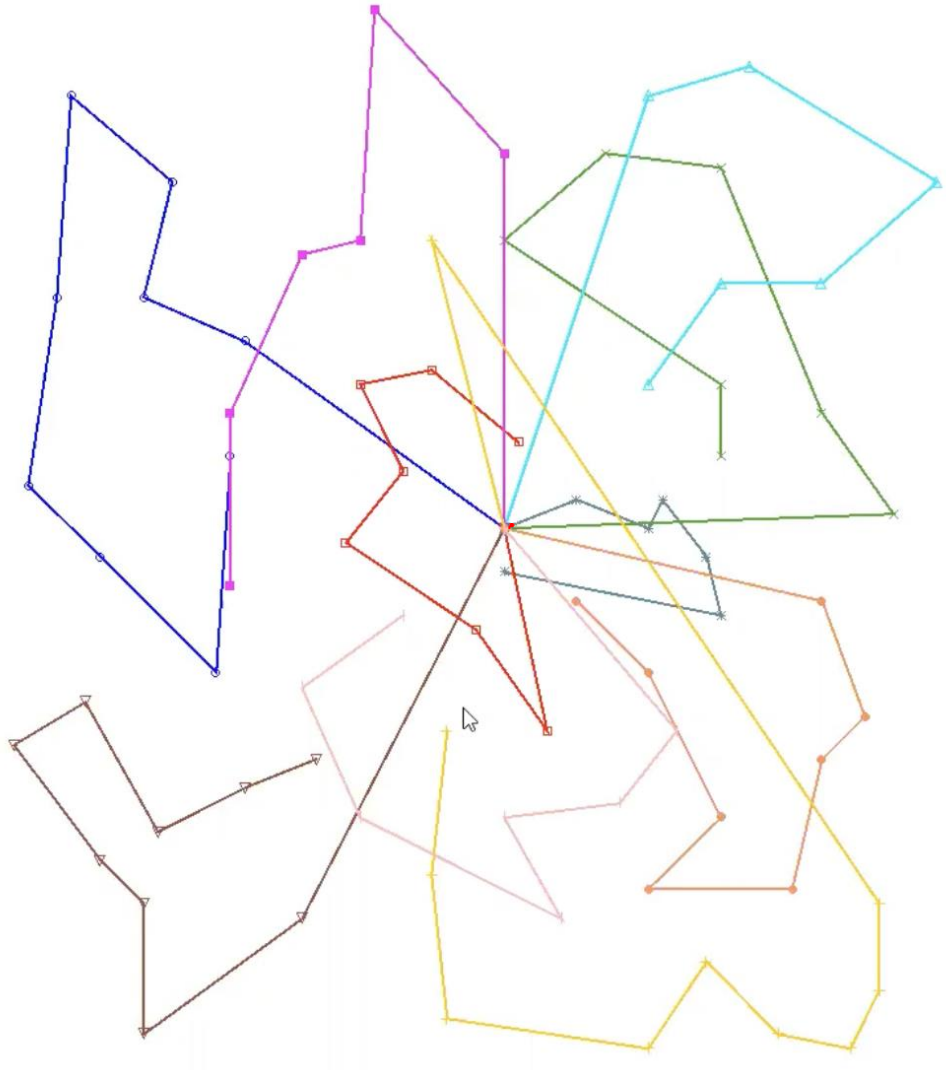
Regret



Regret



Regret

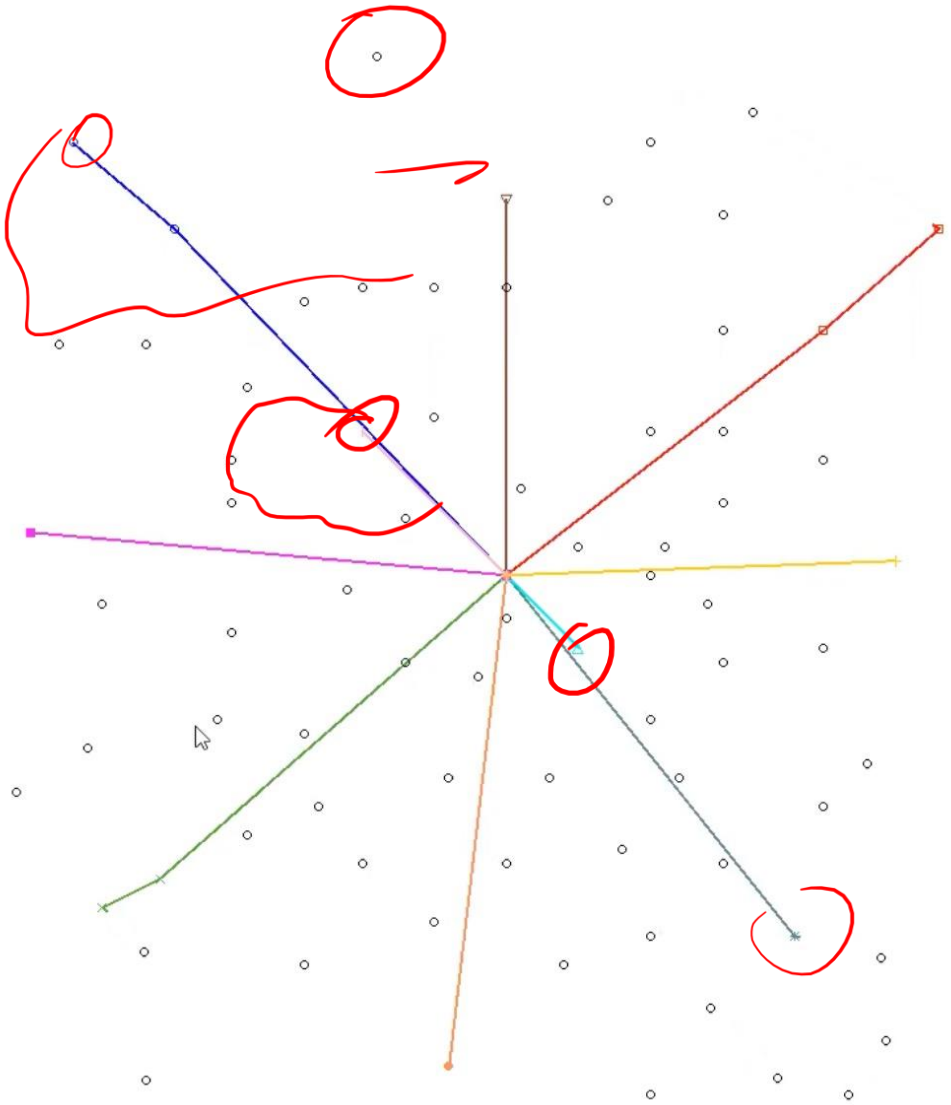


Regret

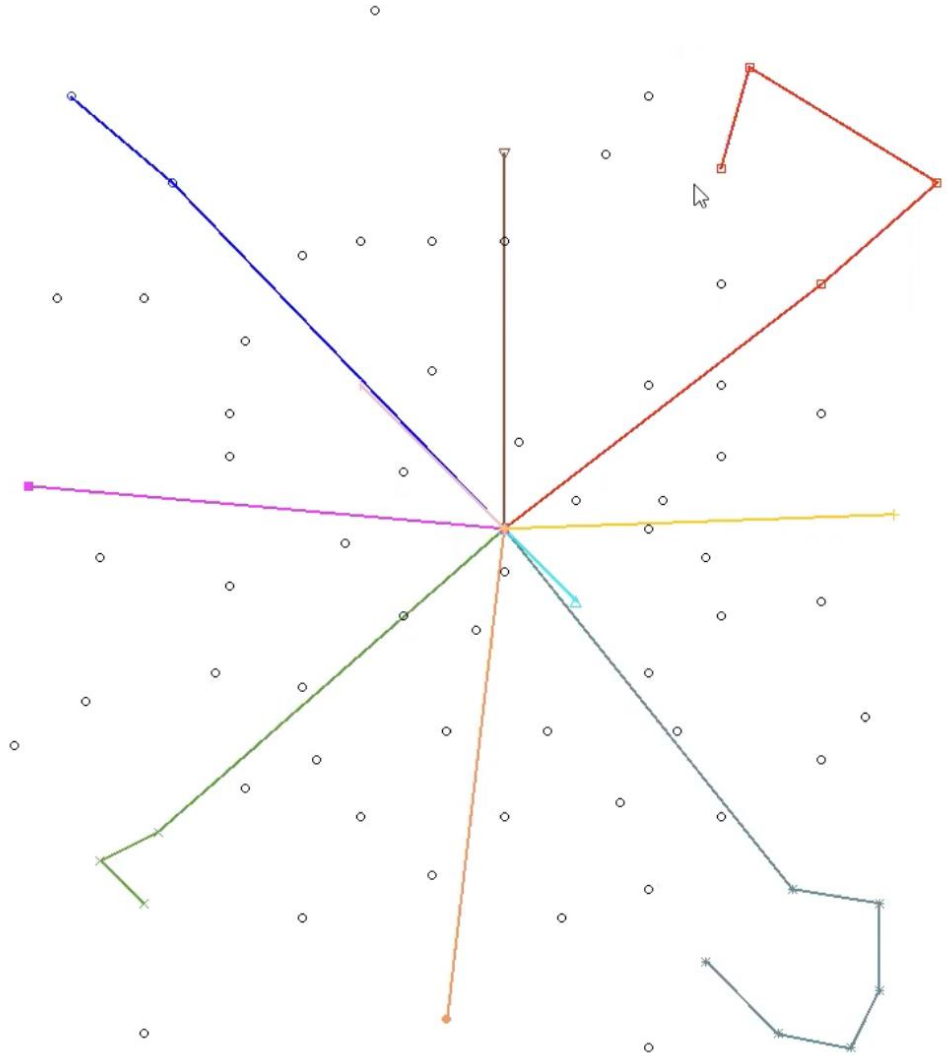
- And a little more structure…
- Regret works best when some structure is already present
- Use Seeds

- Seed 1: The customer with max (dist to depot)
- Seed 2: The customer with max (dist to Seed 1)
- Seed 3: The customer with max (dist to Seed 1 + dist to Seed 2)
- Seed 4: The customer with max (dist to Seed 1 + dist to Seed 2 + dist to Seed 3)
- Etc

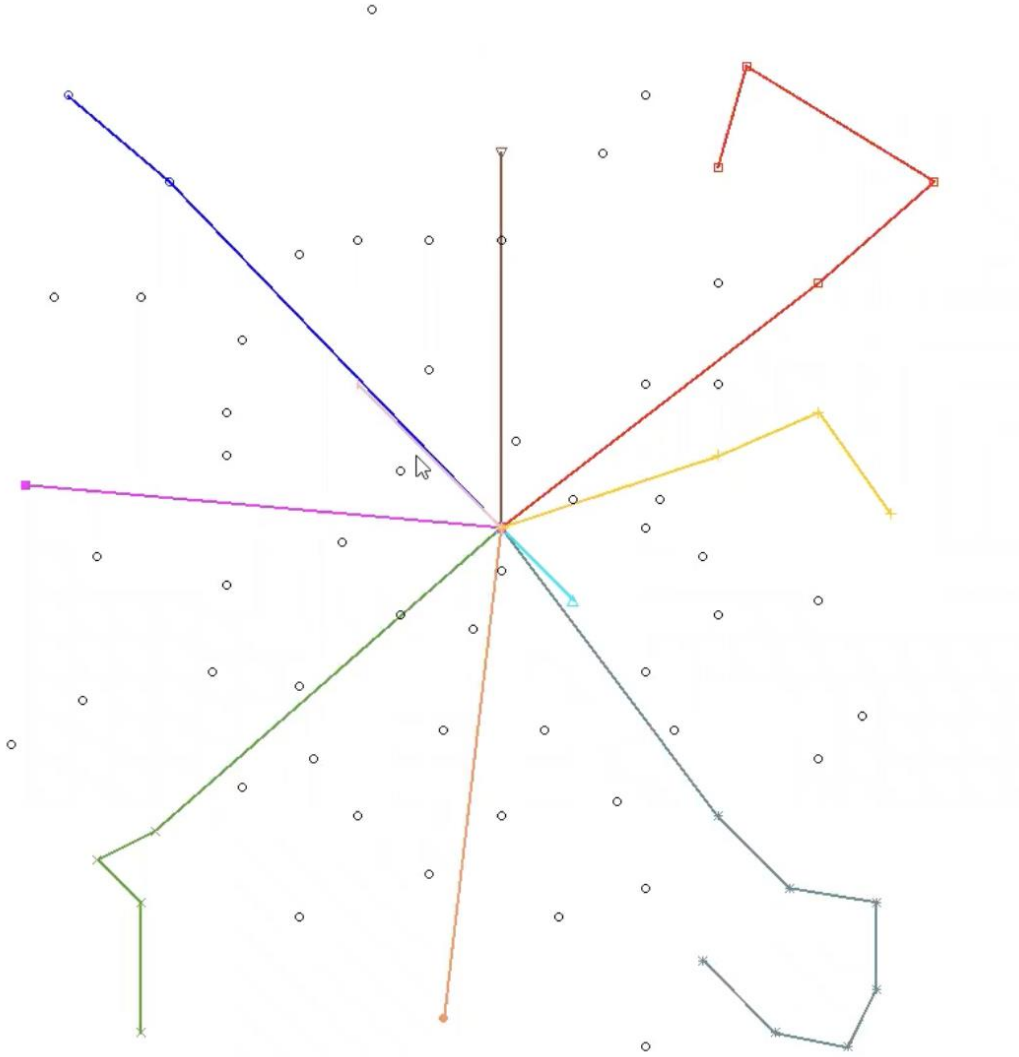
Regret with seeds



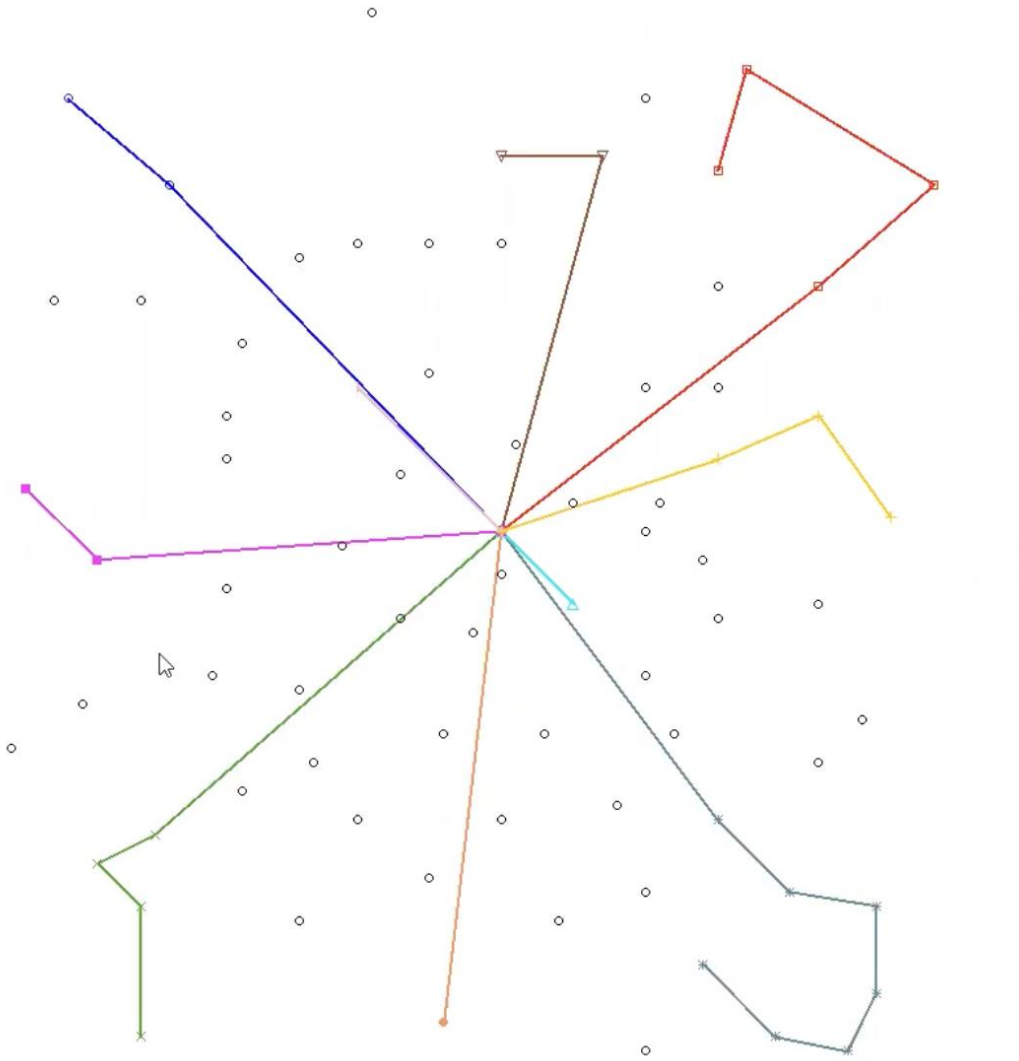
Regret with seeds



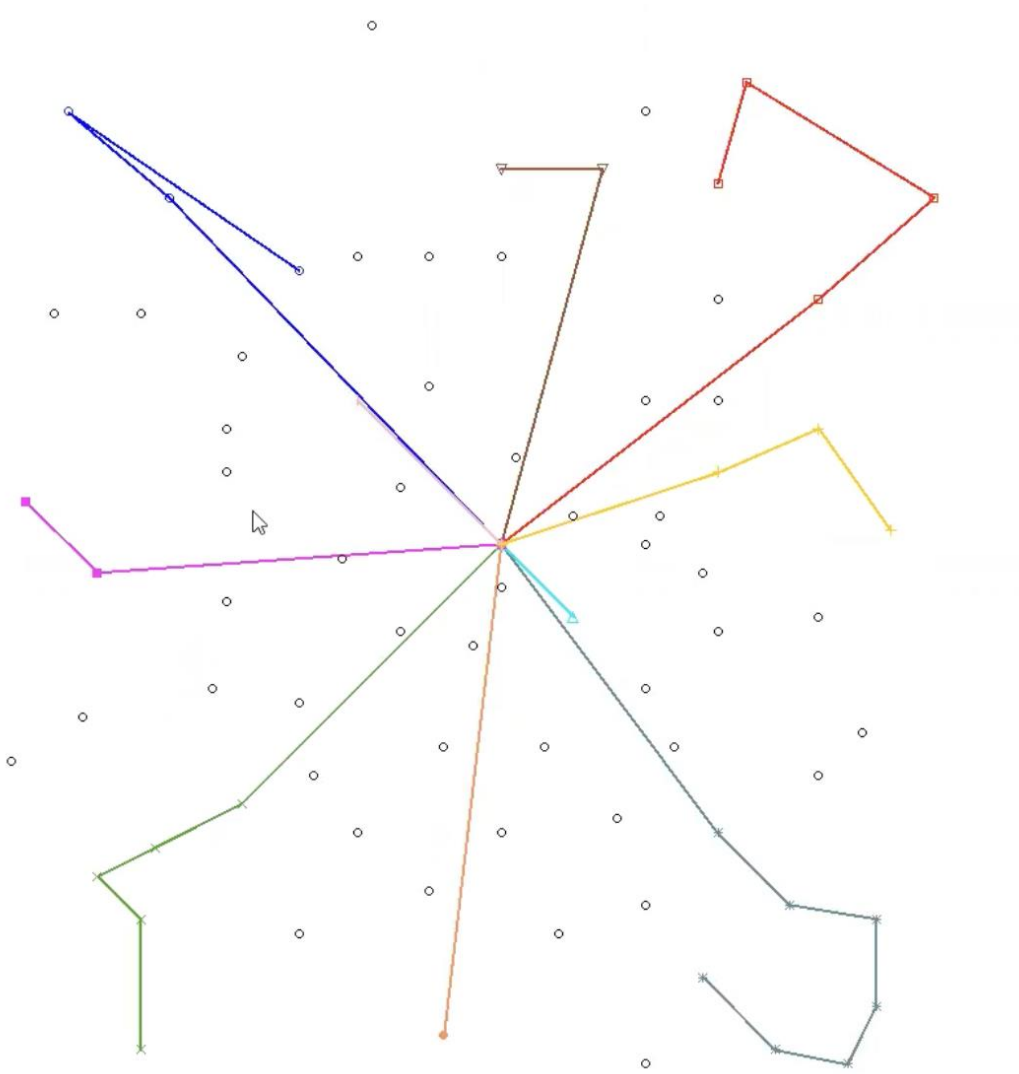
Regret with seeds



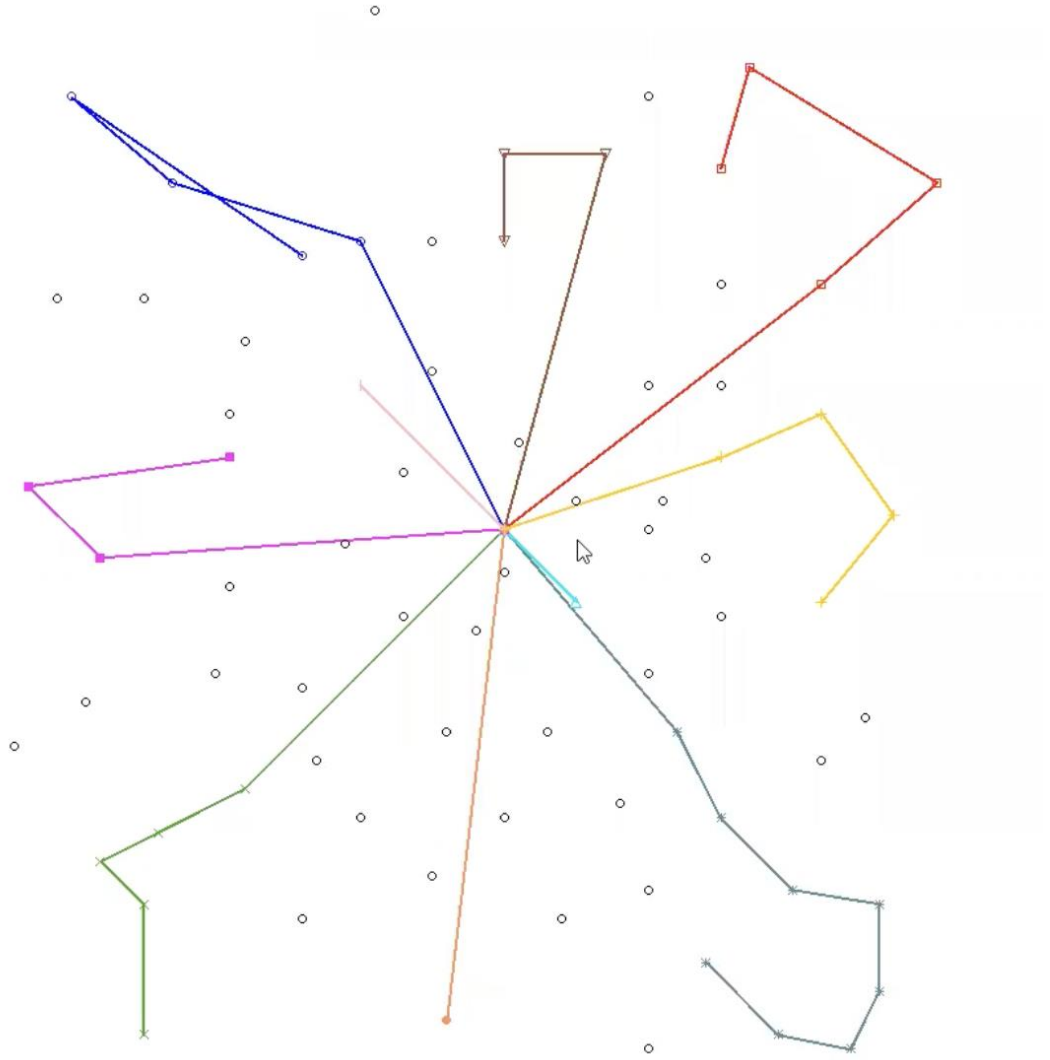
Regret with seeds



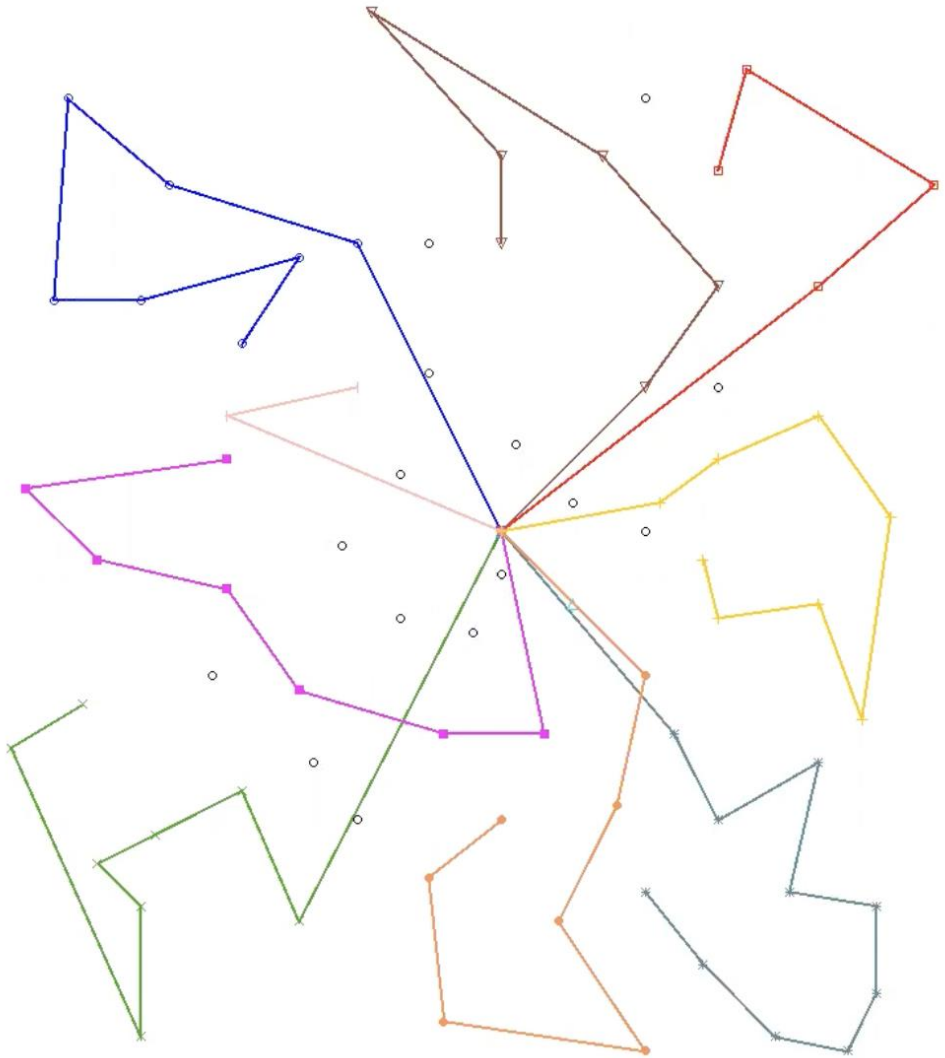
Regret with seeds



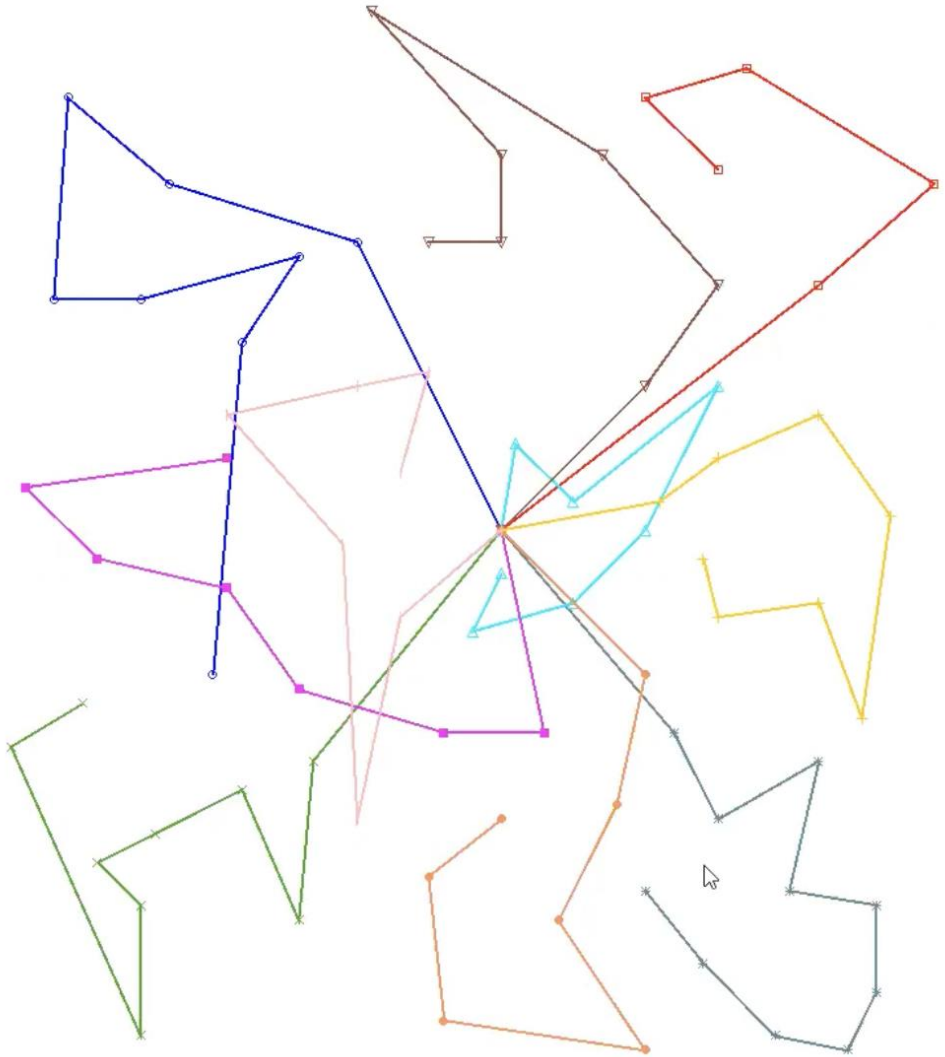
Regret with seeds



Regret with seeds

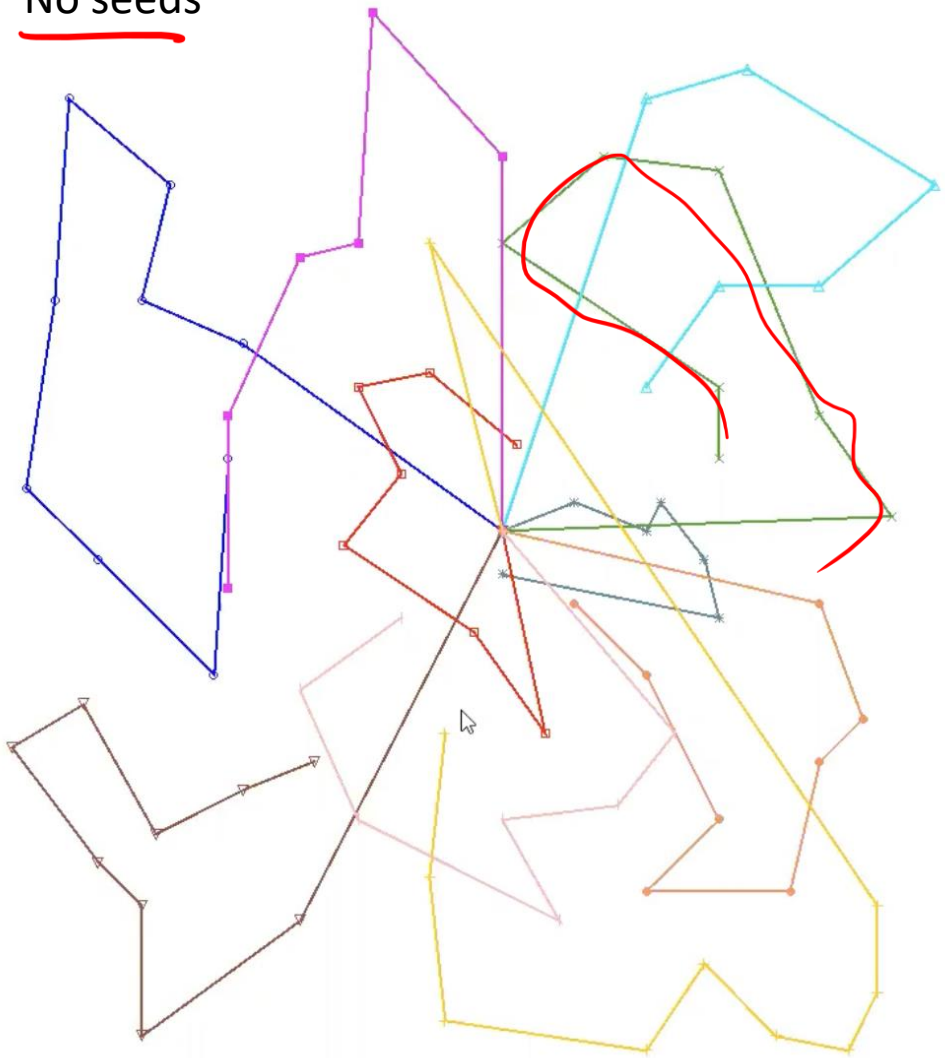


Regret with seeds

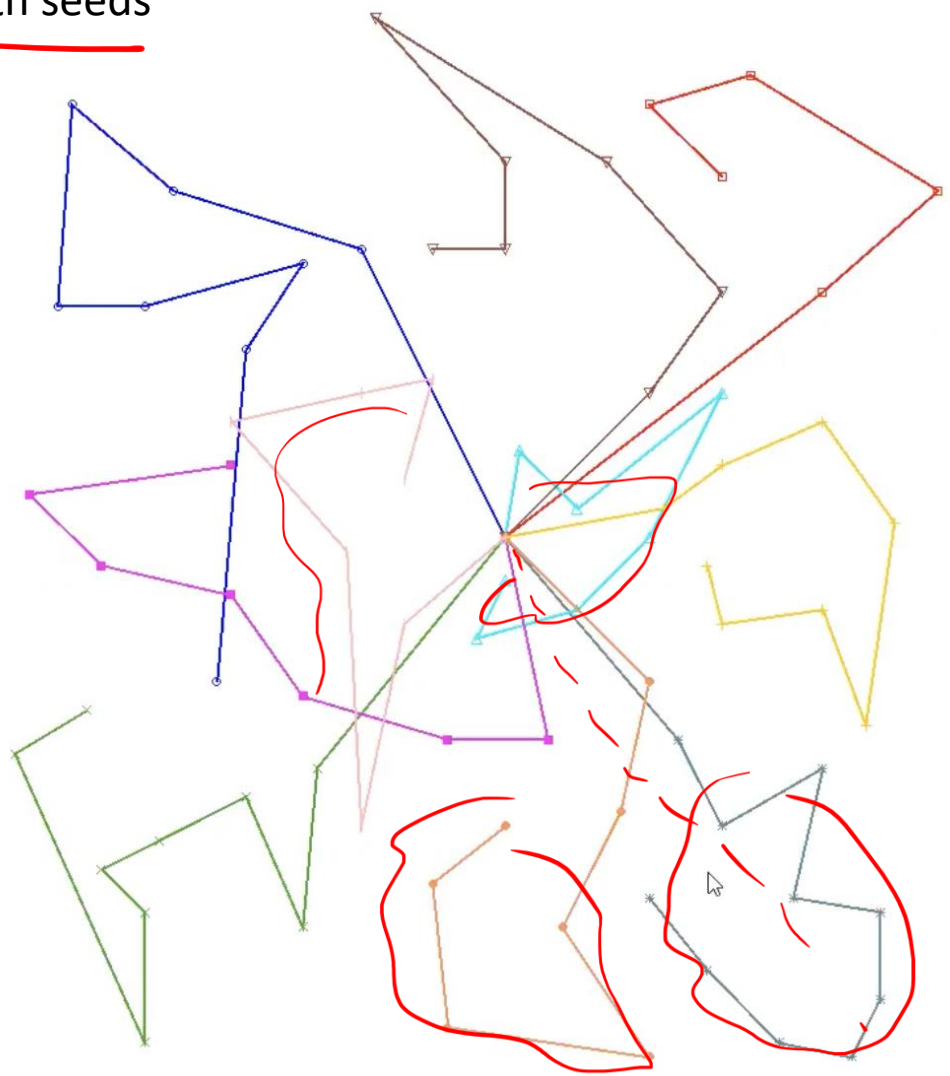


Regret with seeds

No seeds



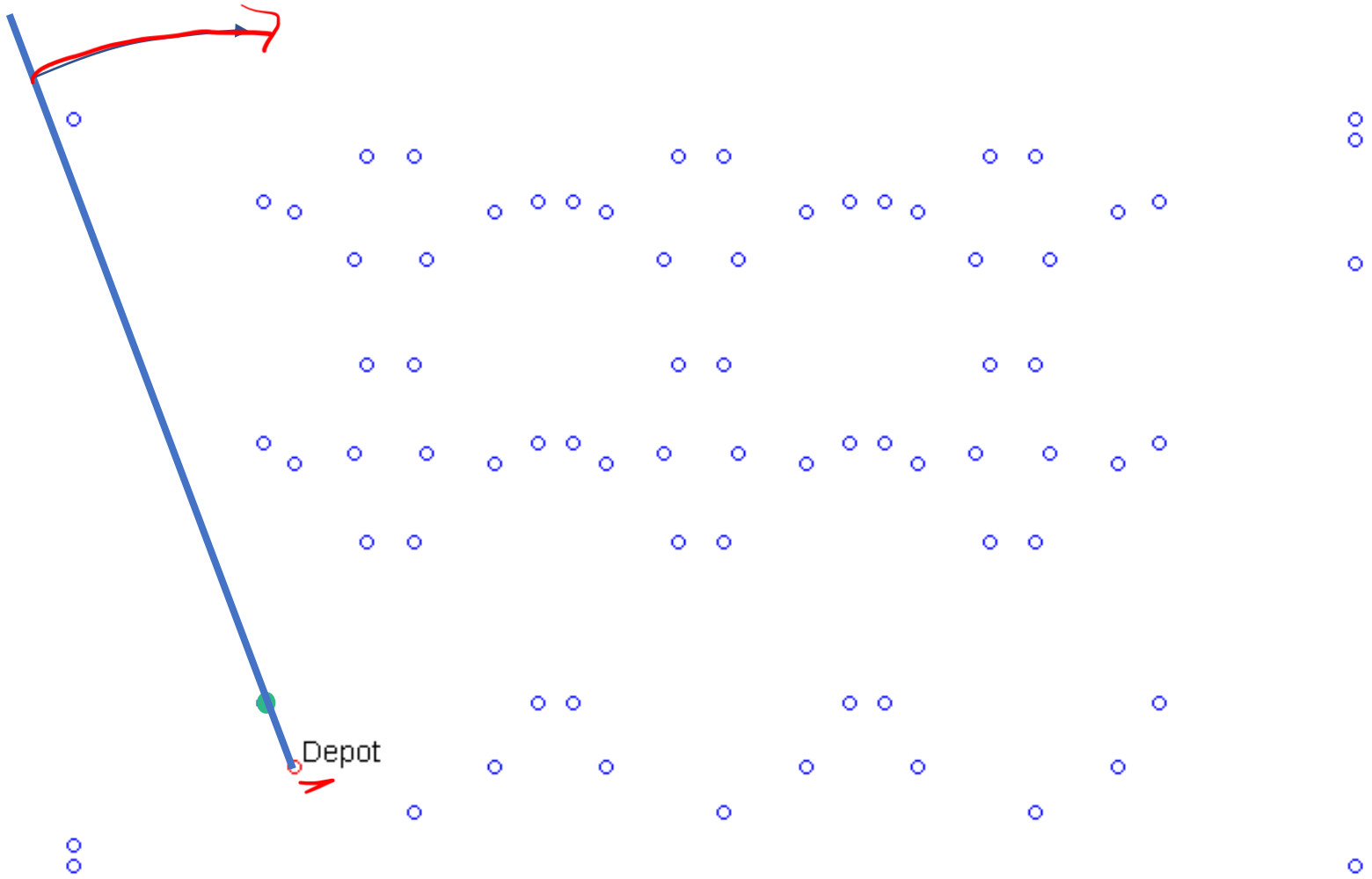
With seeds



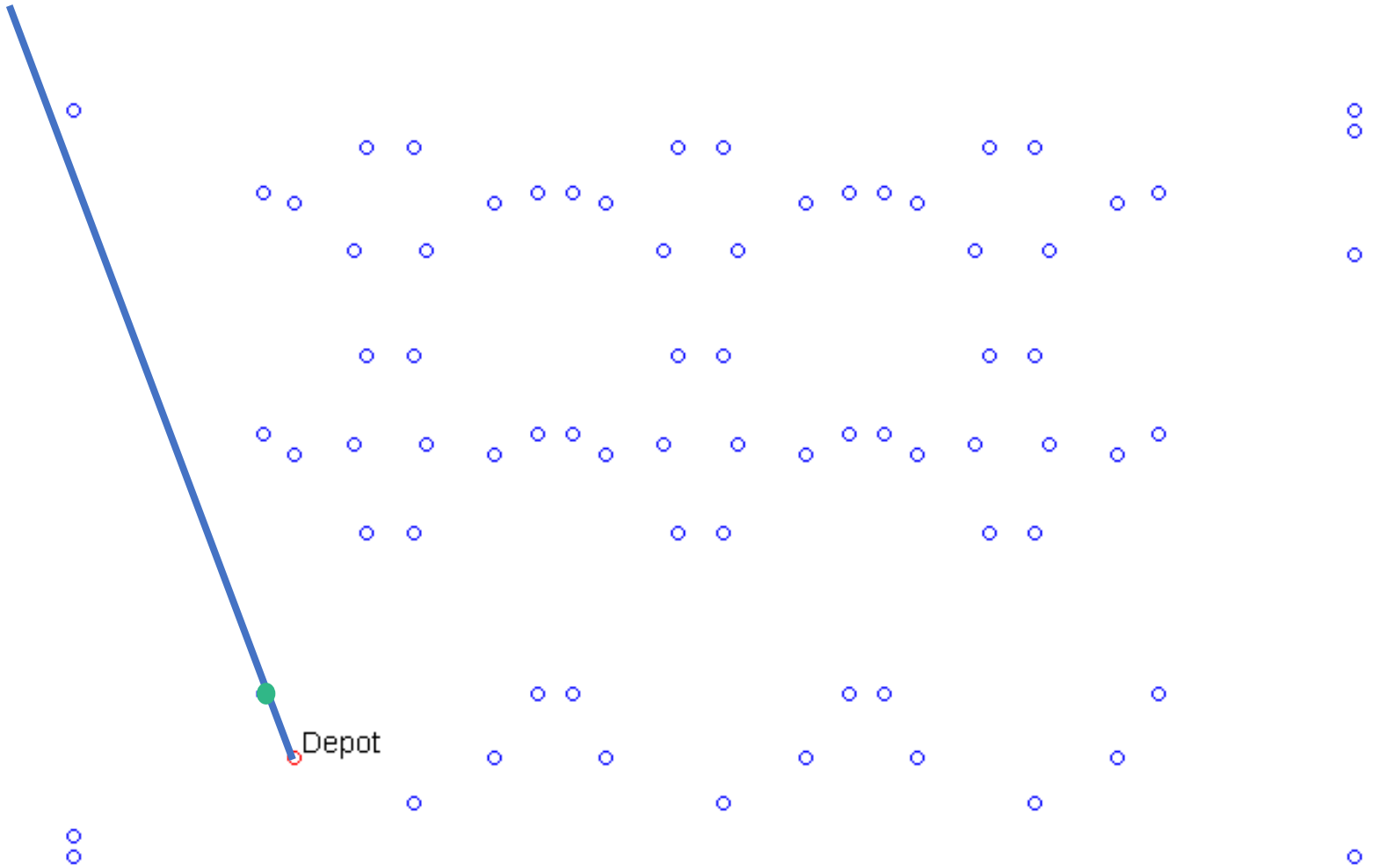
Other construction methods (for the VRP)

Bespoke methods

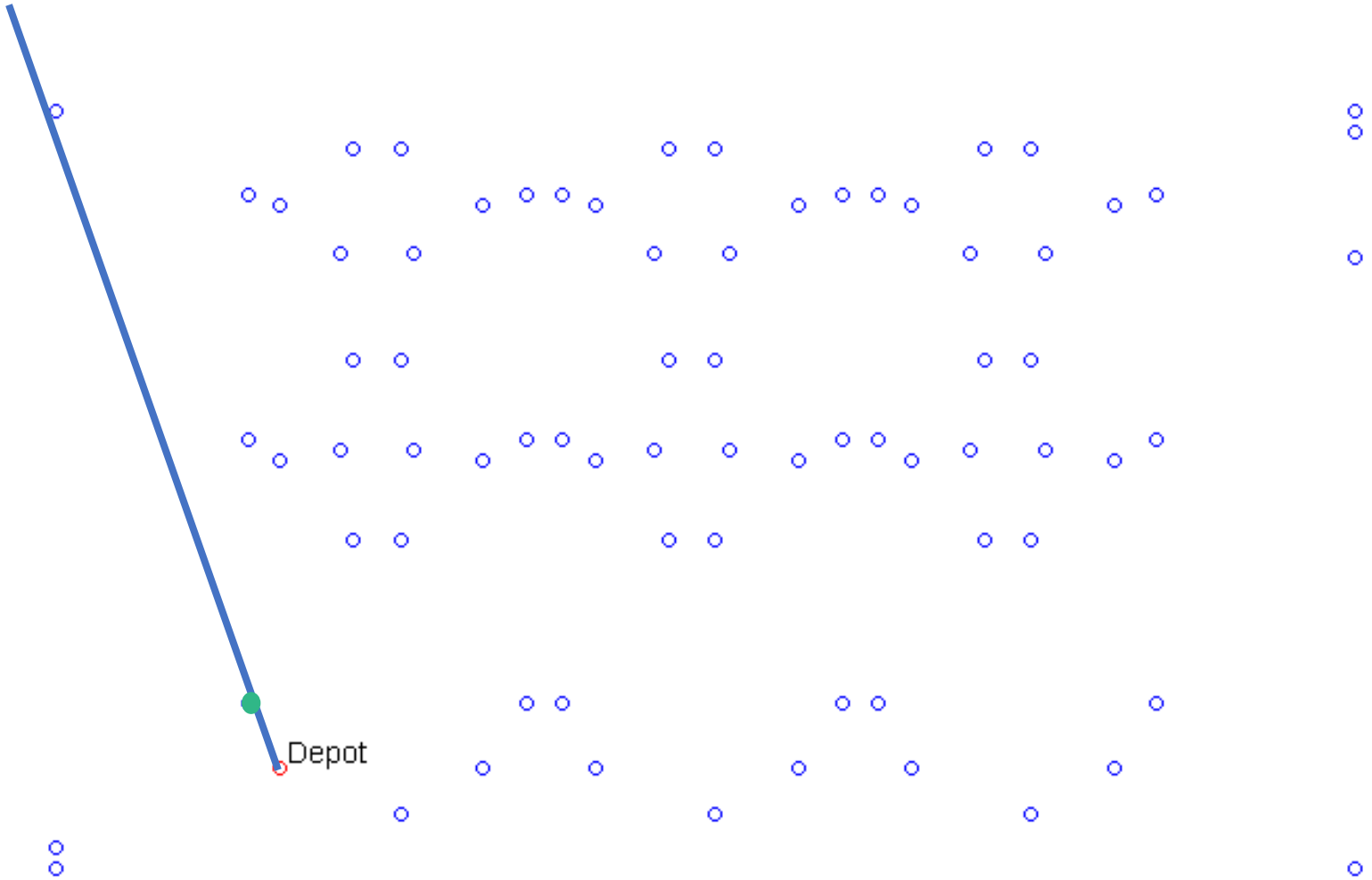
- Eg “Sweep”



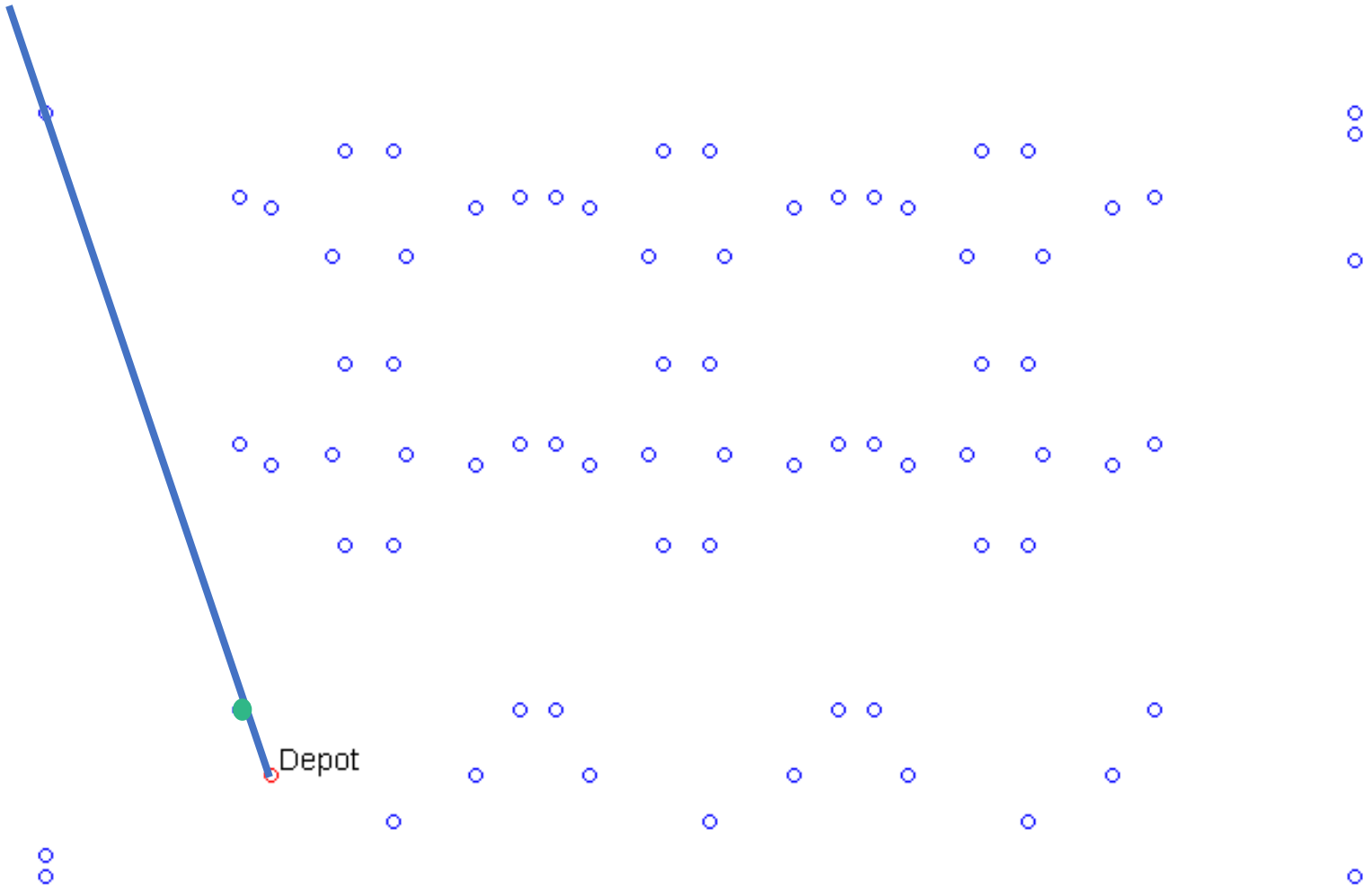
Other construction methods (for the VRP)



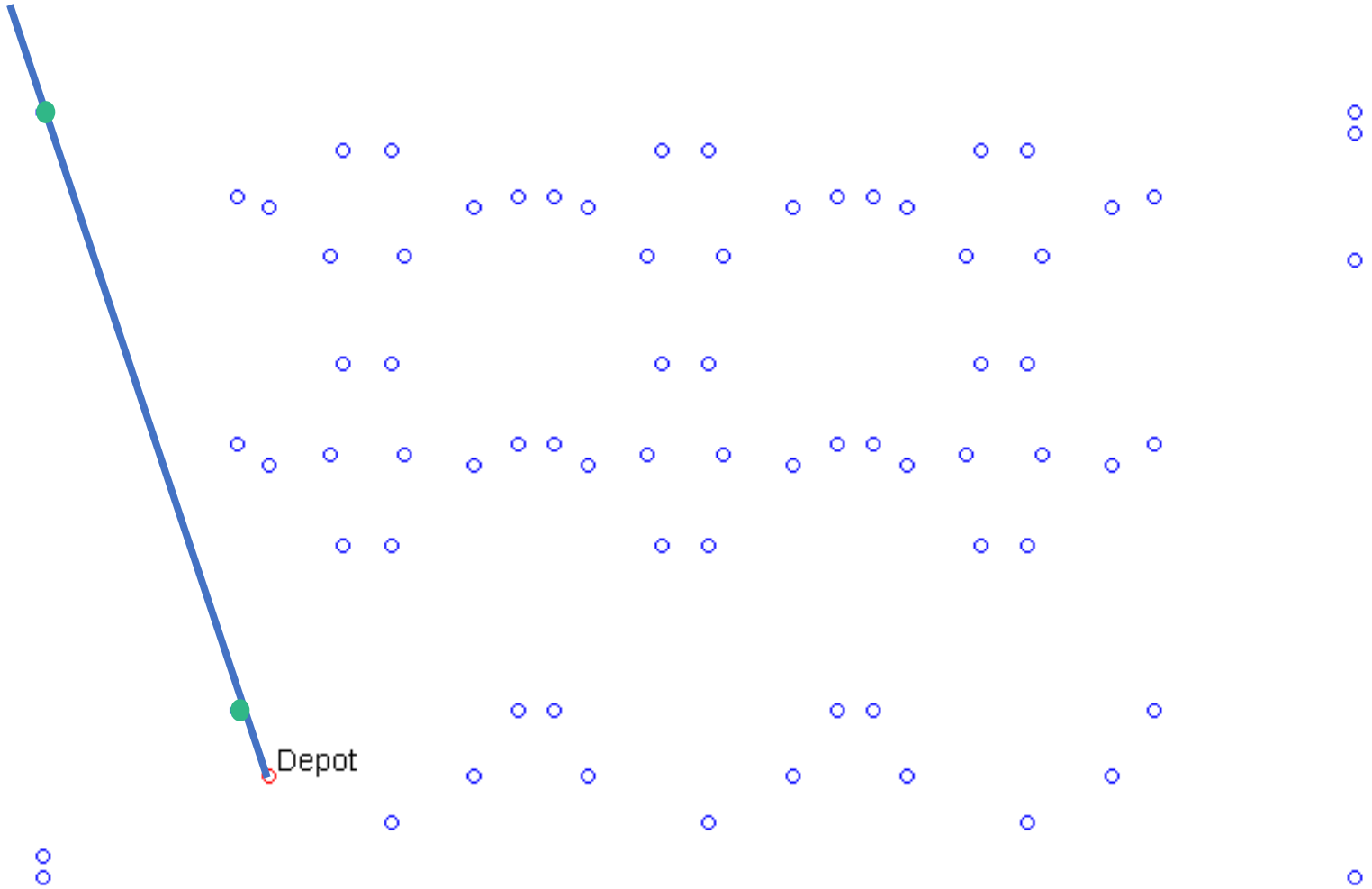
Other construction methods (for the VRP)



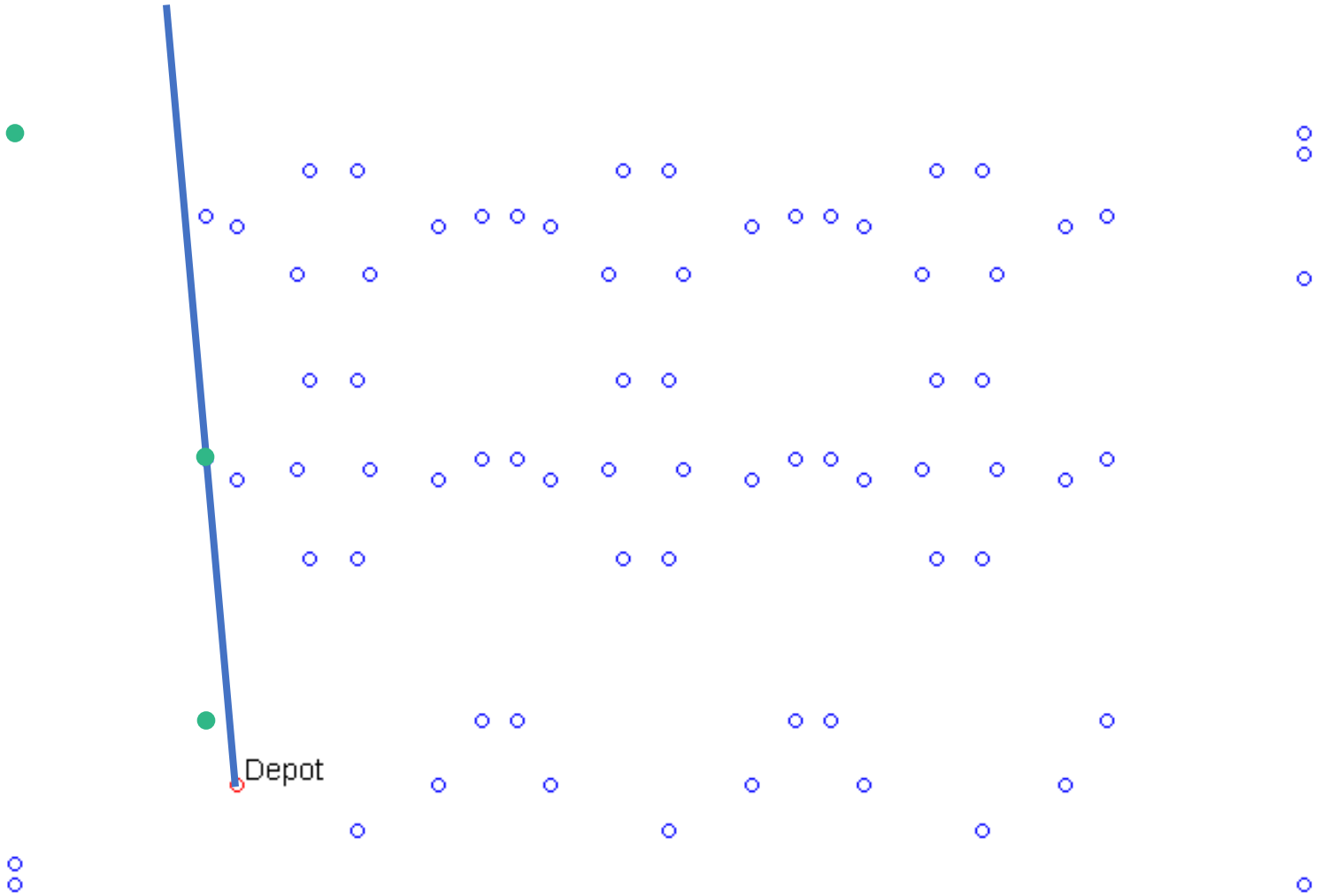
Other construction methods (for the VRP)



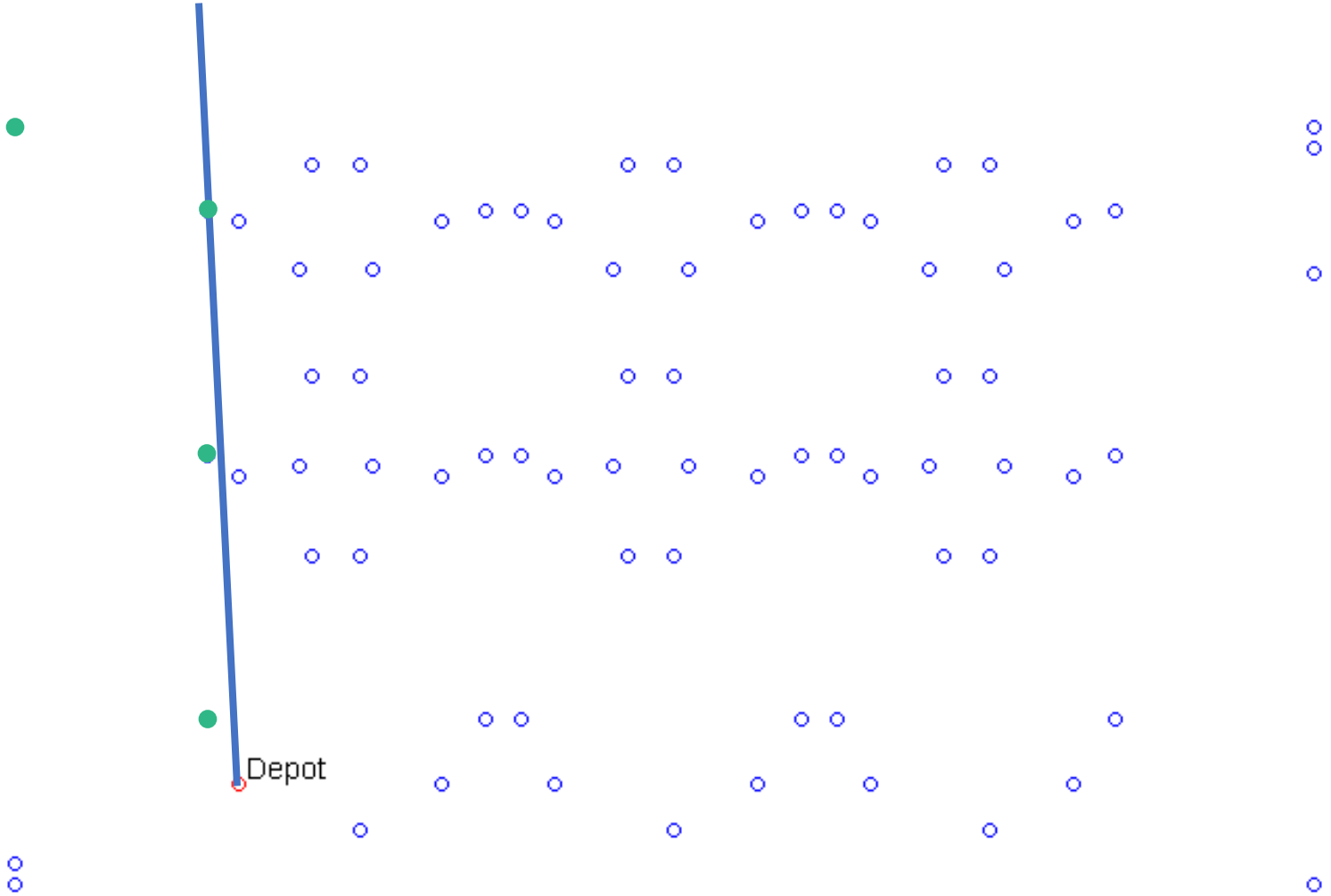
Other construction methods (for the VRP)



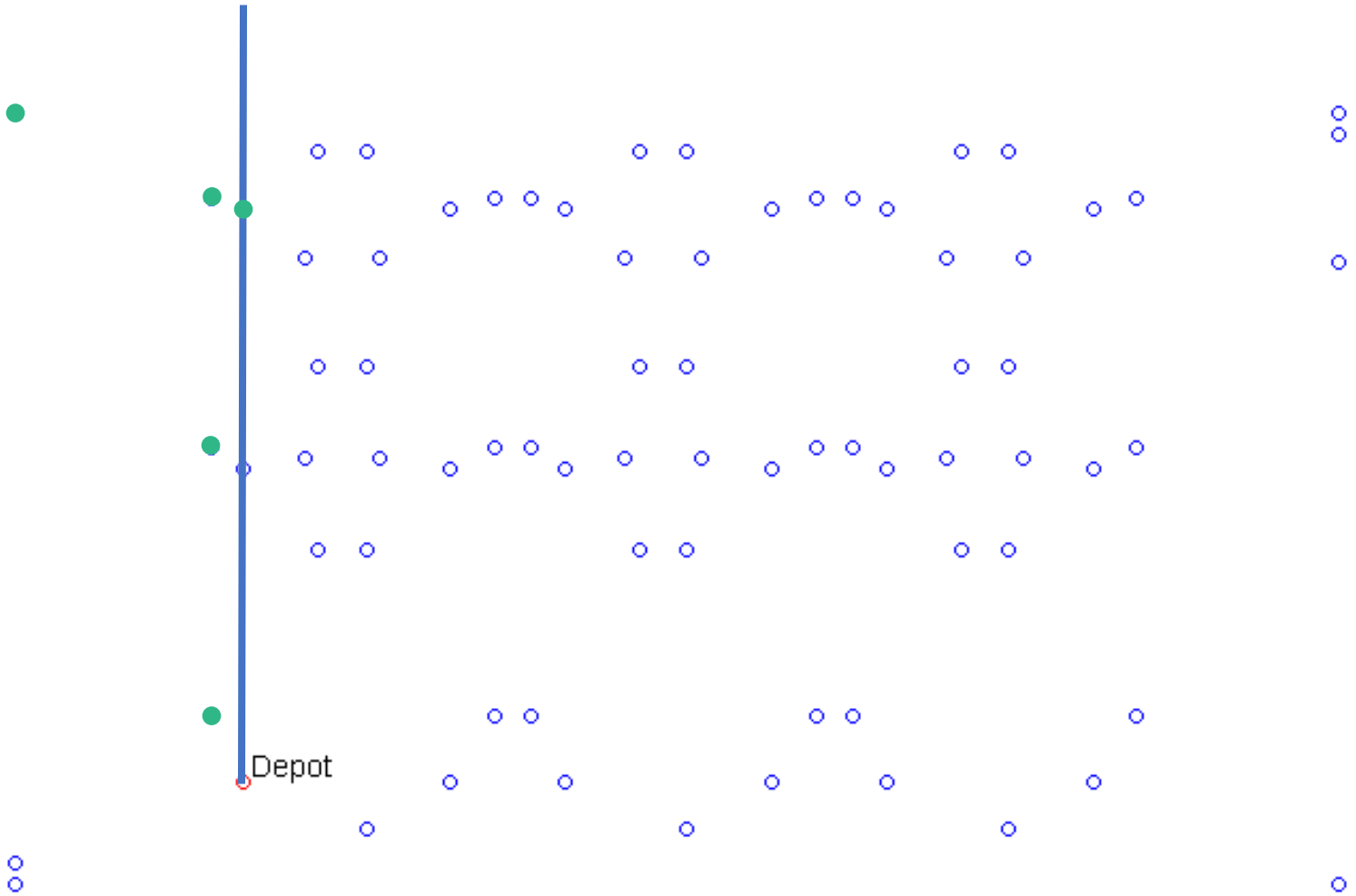
Other construction methods (for the VRP)



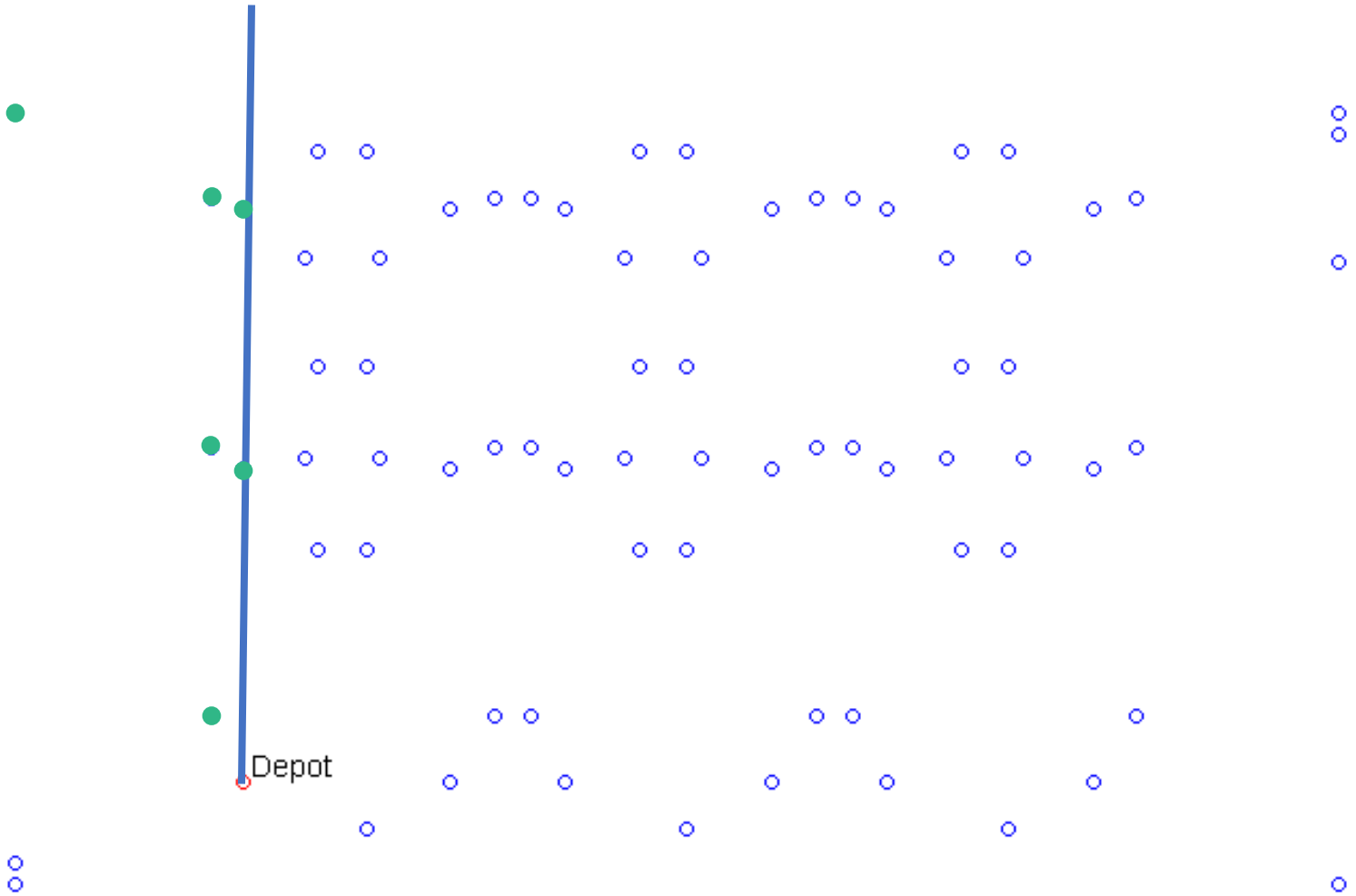
Other construction methods (for the VRP)



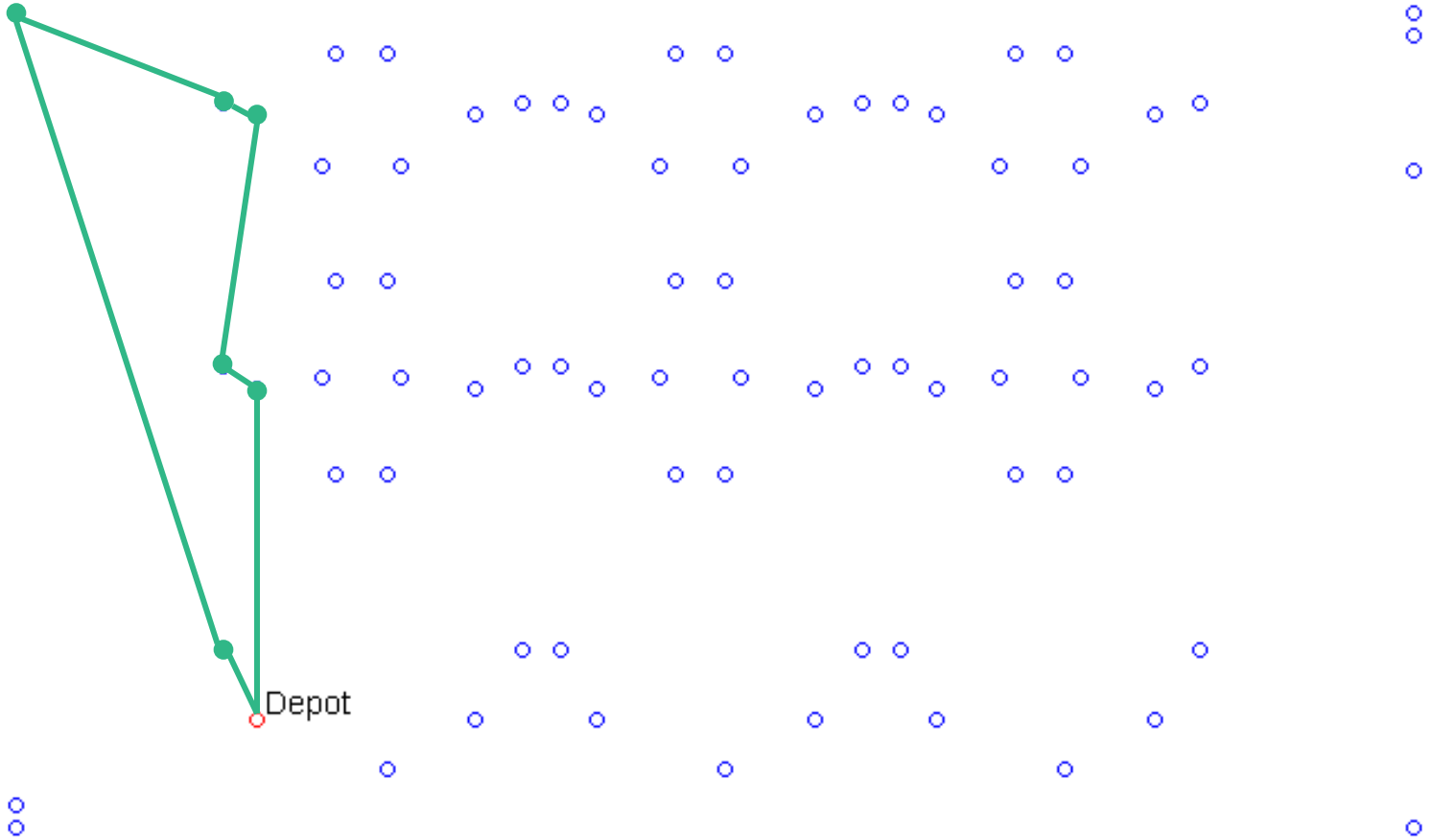
Other construction methods (for the VRP)



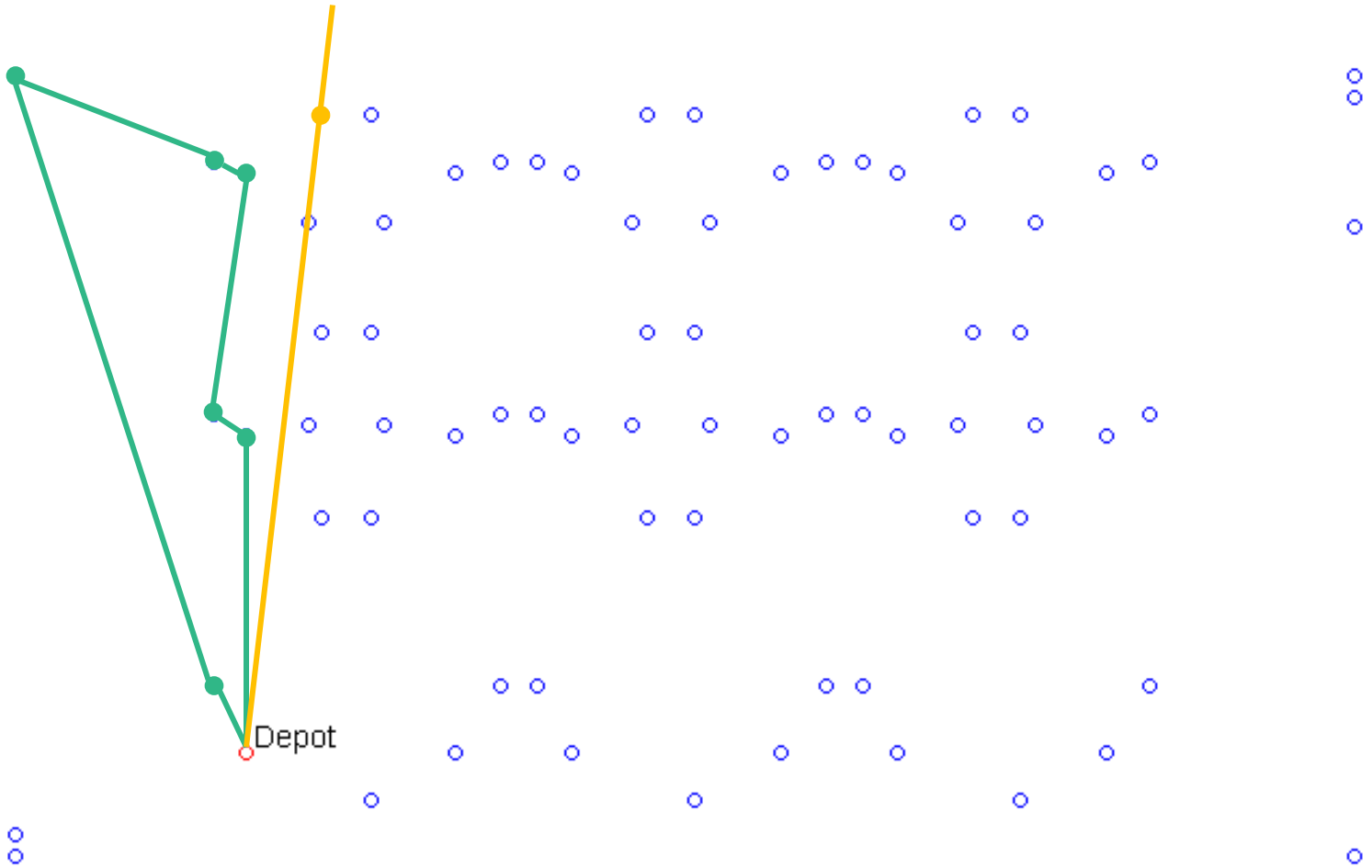
Other construction methods (for the VRP)



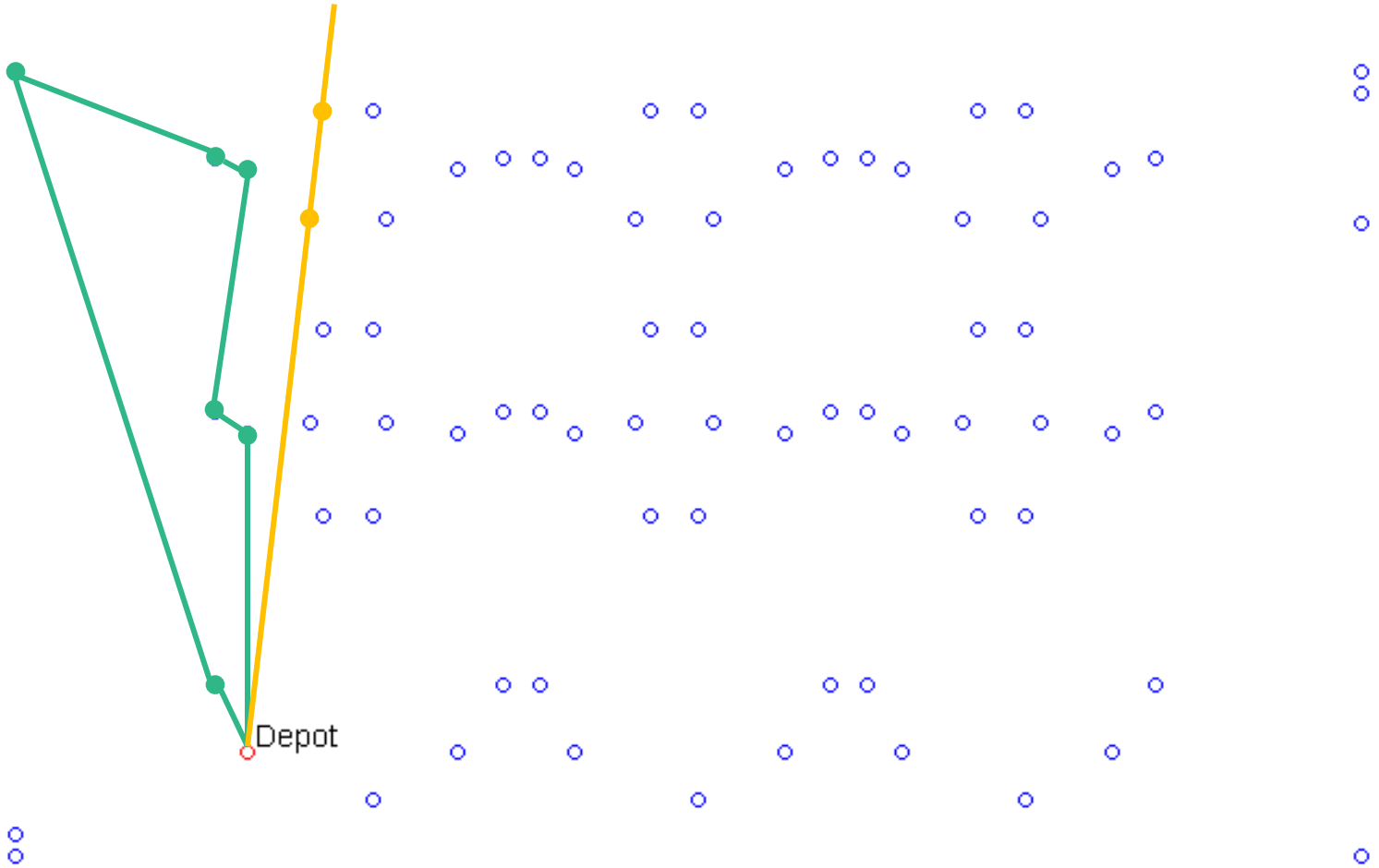
Other construction methods (for the VRP)



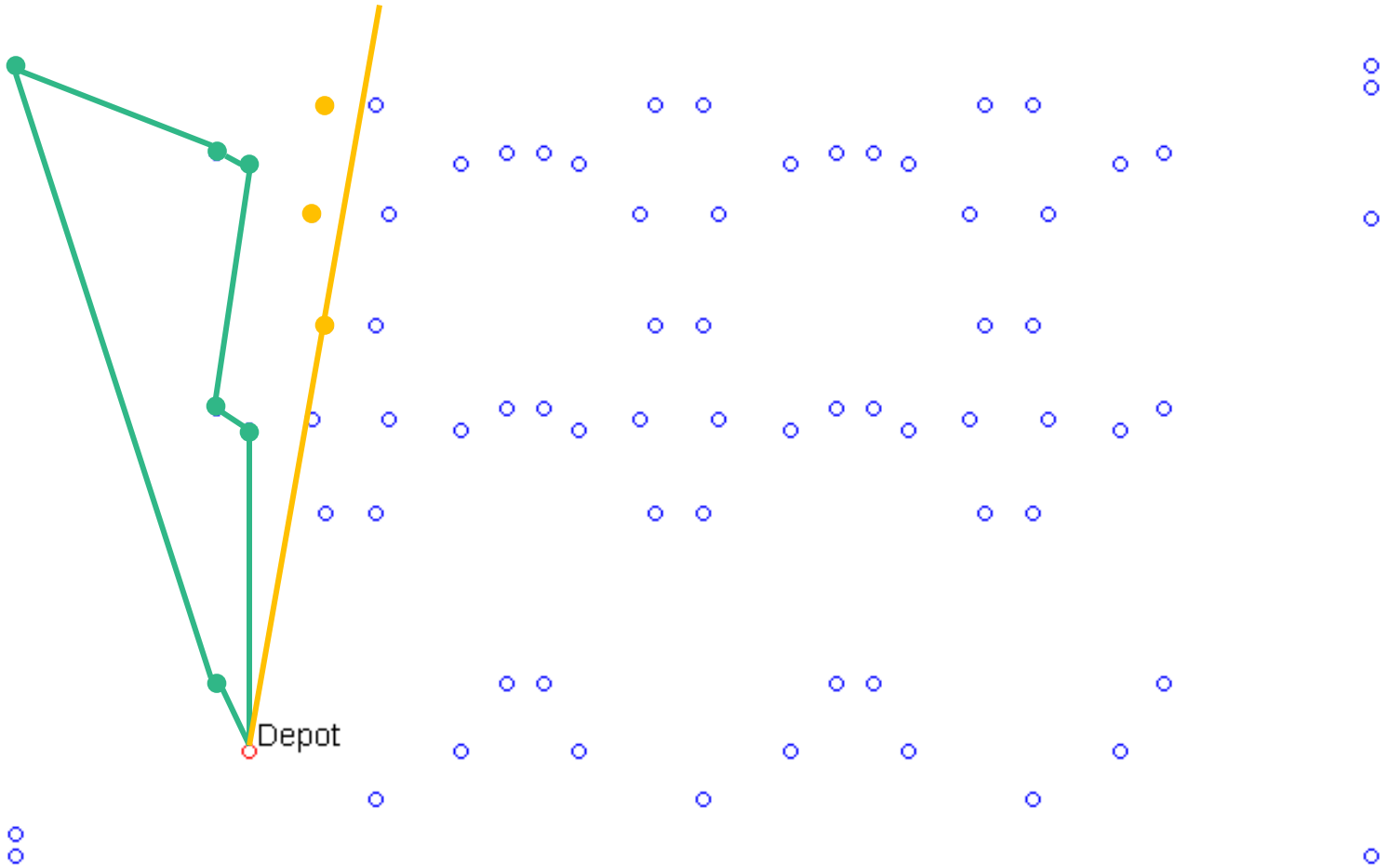
Other construction methods (for the VRP)



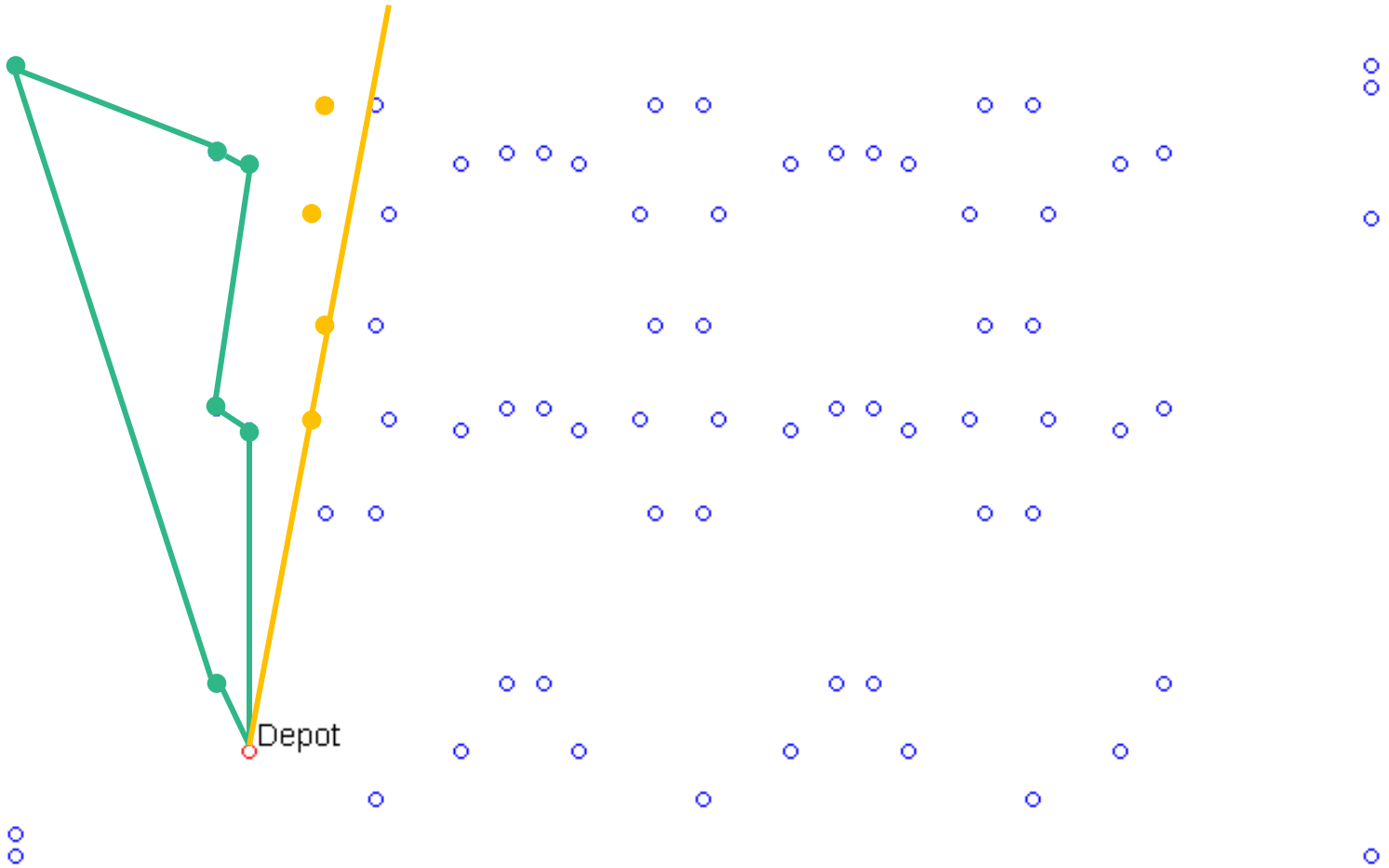
Other construction methods (for the VRP)



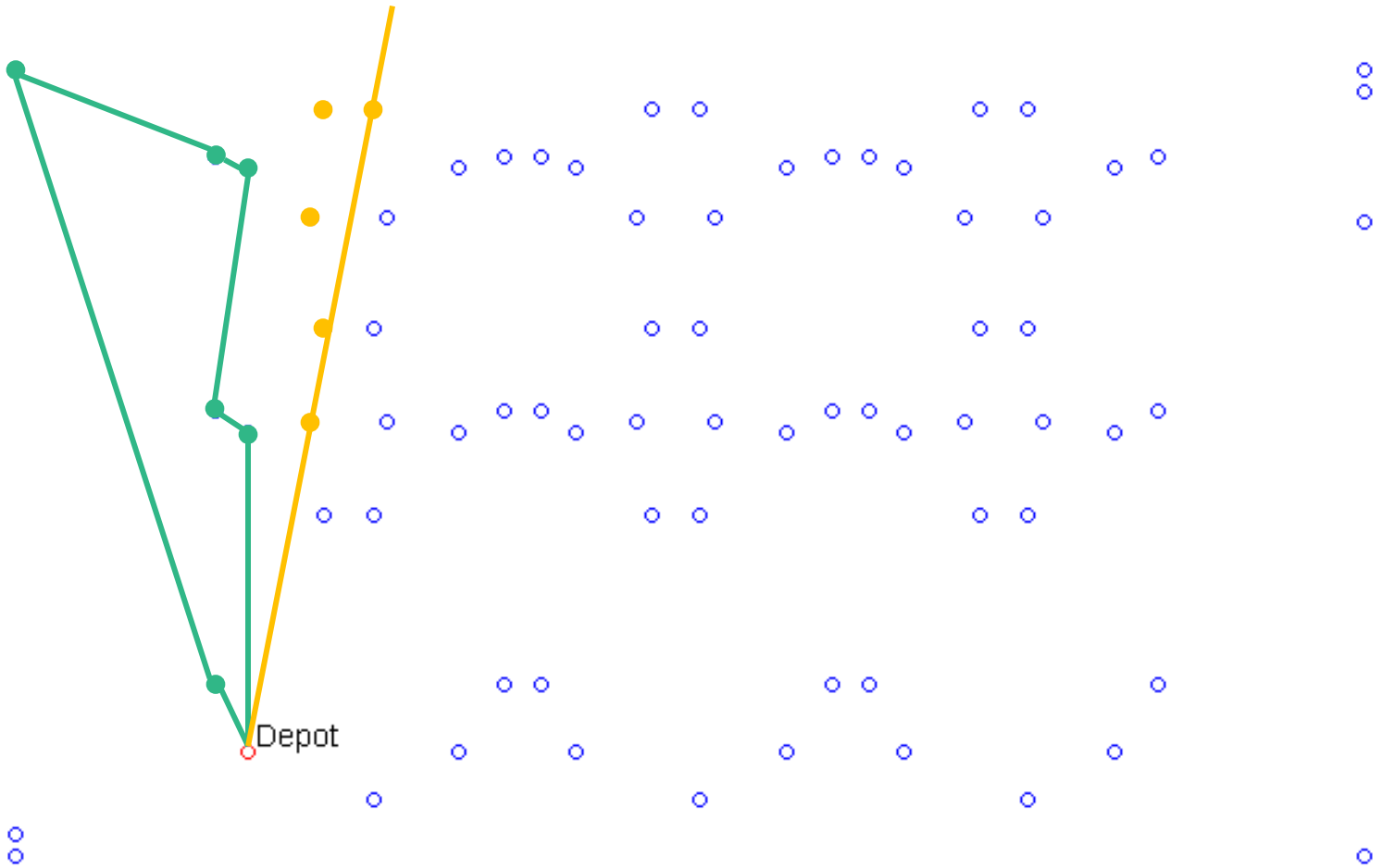
Other construction methods (for the VRP)



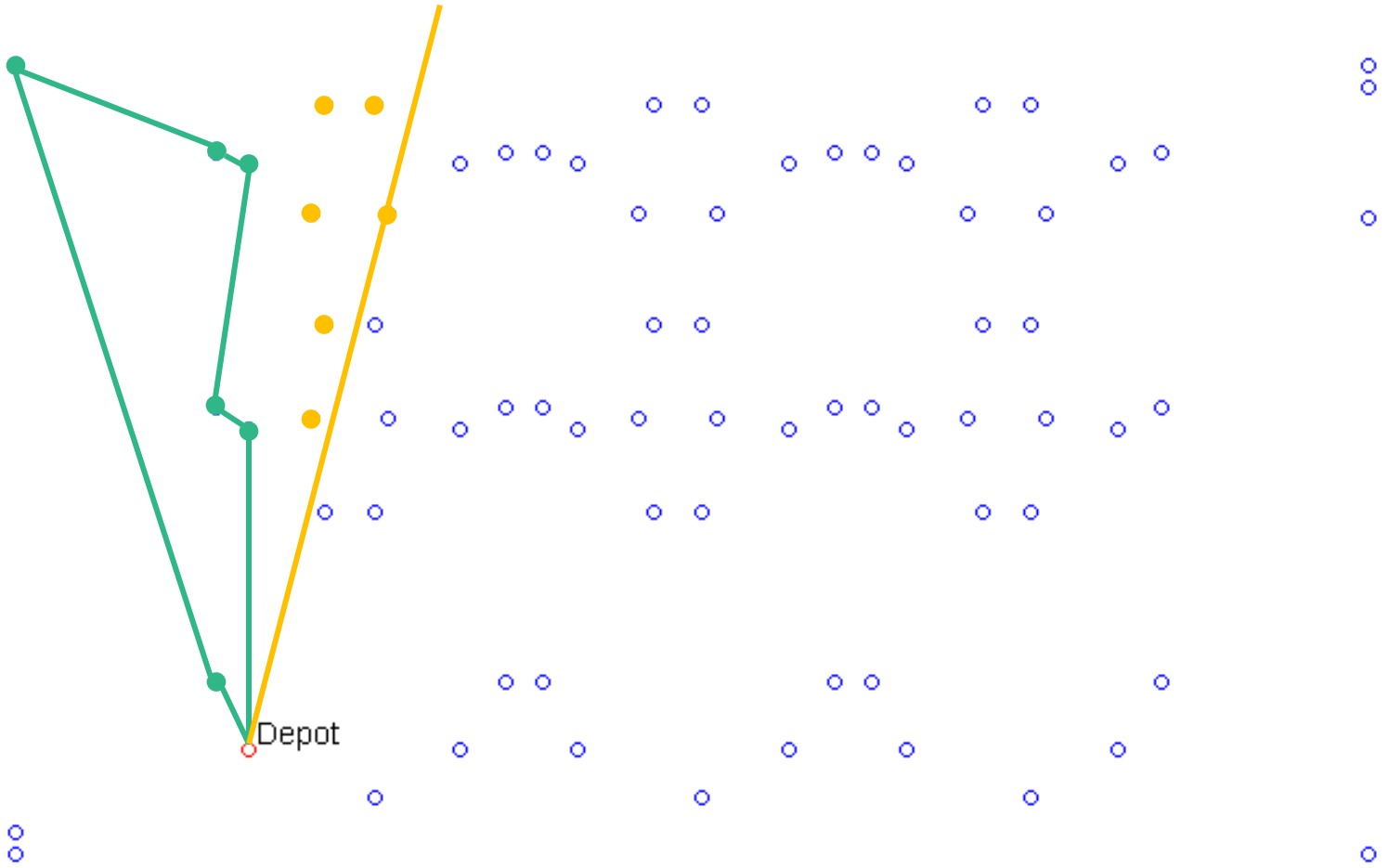
Other construction methods (for the VRP)



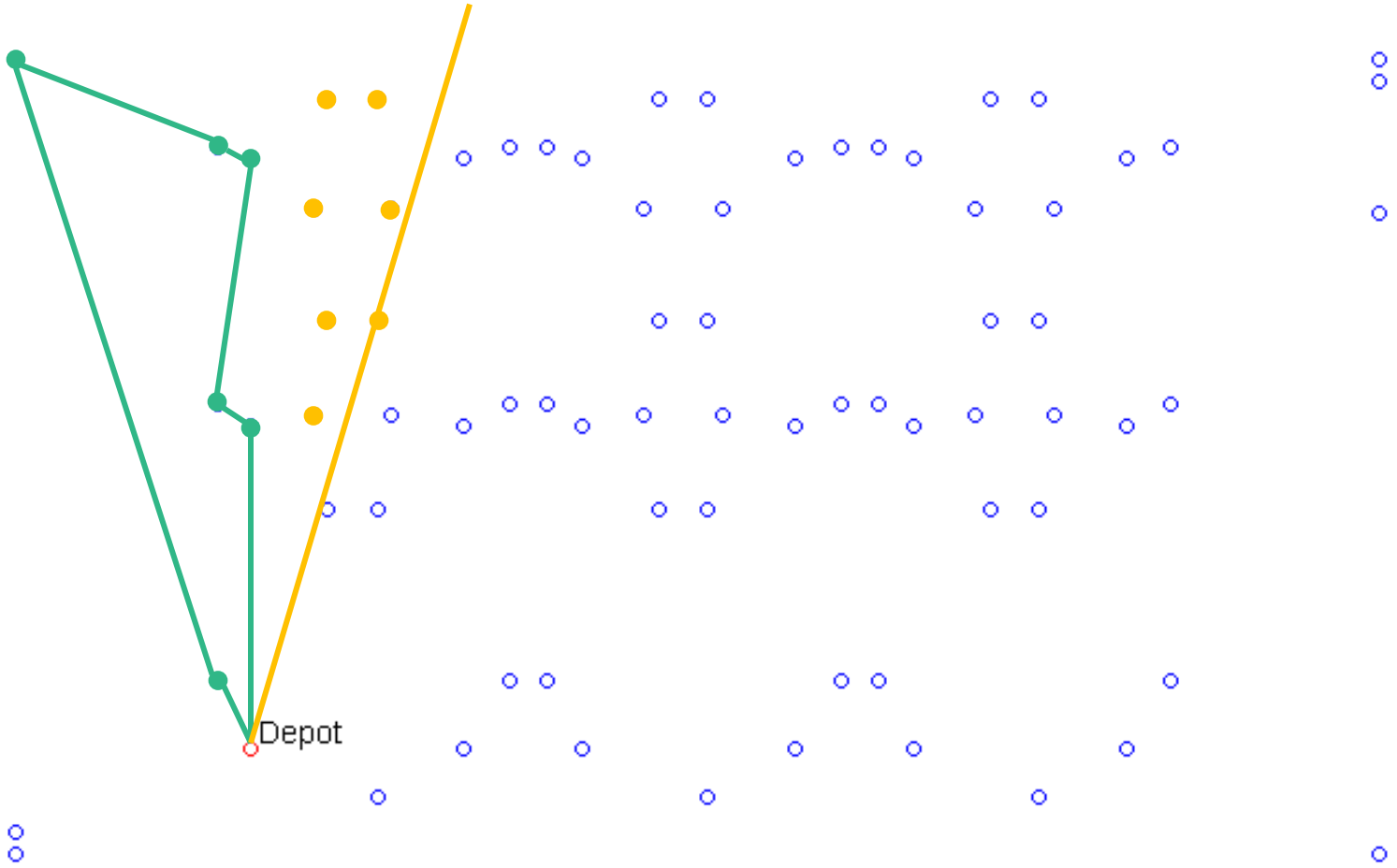
Other construction methods (for the VRP)



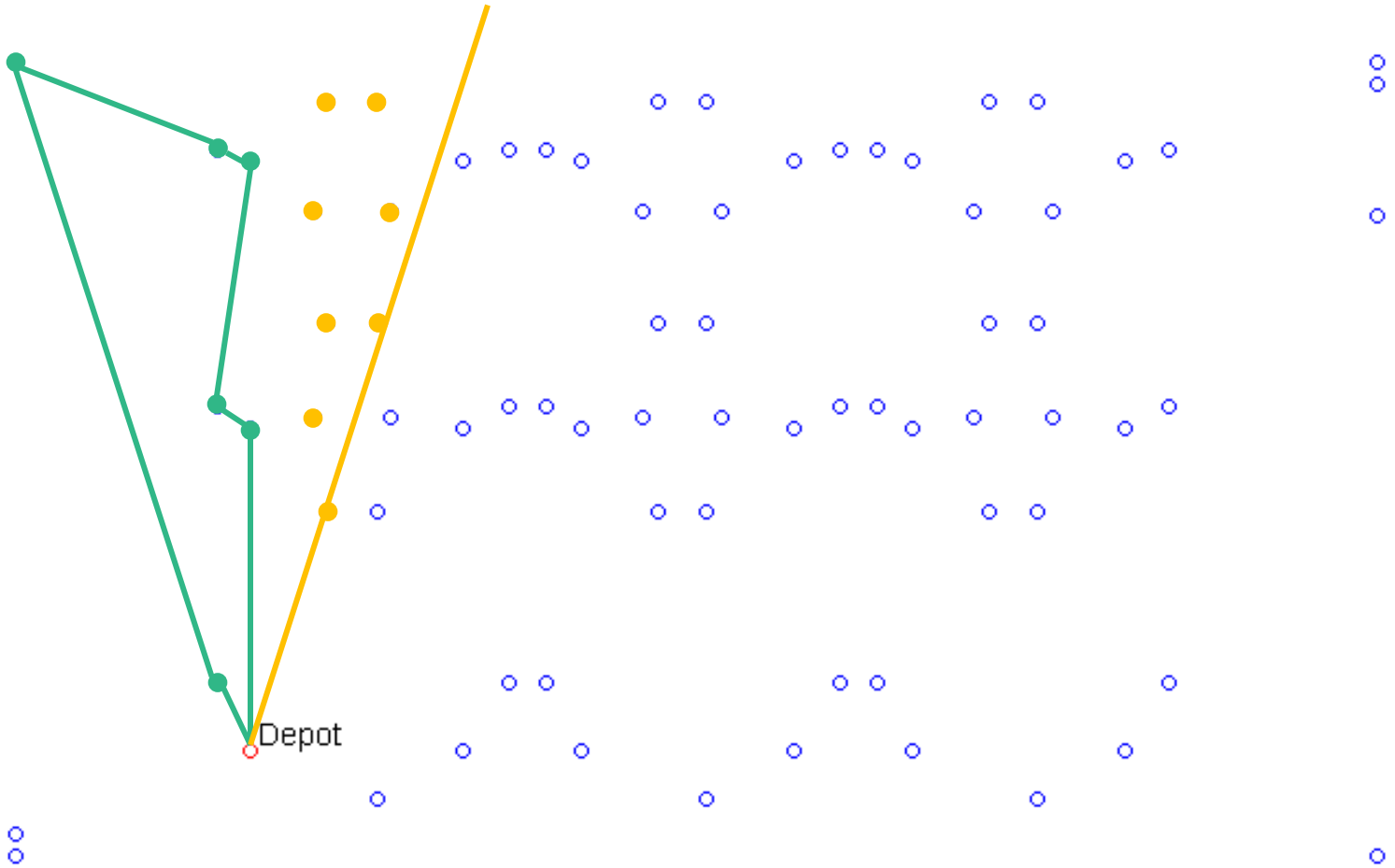
Other construction methods (for the VRP)



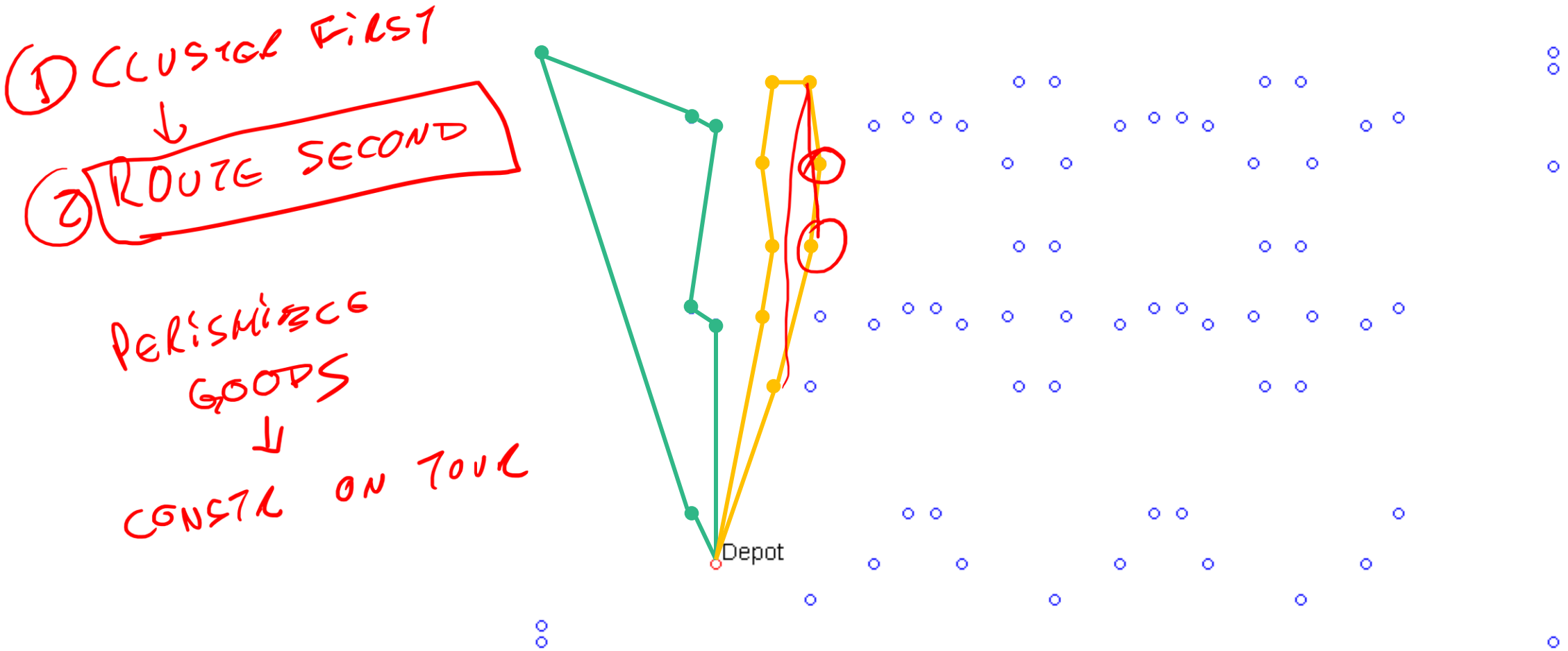
Other construction methods (for the VRP)



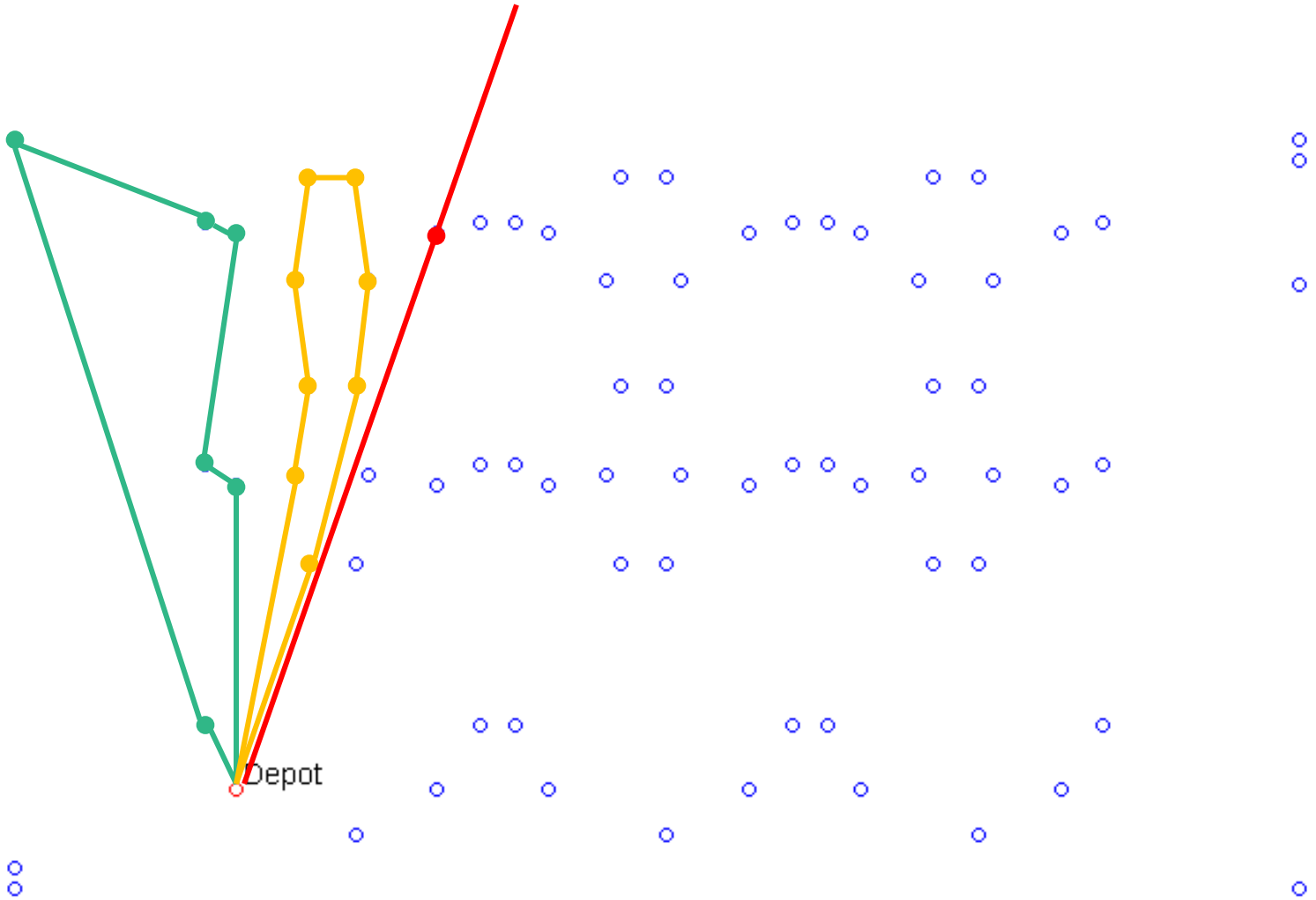
Other construction methods (for the VRP)



Other construction methods (for the VRP)



Other construction methods (for the VRP)



Other construction methods

3D Packing – Knapsack in 3D

- Working with a large mining co.
- They truck stuff from distributions centres near Perth to their mines
- Lots of stuff

3D Packing – Knapsack in 3D

- Working with a large mining co.
- They truck stuff from distributions centres near Perth to their mines
- Lots of stuff
- Lots and lots of stuff

The problem

- The company use 3PLs (Third-Party Logistics Providers) to move their stuff
- 3PLs are paid by the truck
- 3PLs *say* they pack their trucks well, **but** there is no incentive to do so

The problem

- The company *know* they are being ripped off
- They just don't know how much

- This is a "size of opportunity" project
- Aim: Answer the question

If 3PLs packed their trucks really, really well, how much could we save?

- Answer: $5M – Meh
- Answer: $20M – Let's have a closer look

Problem statement

Given a collection of items and a fleet of trucks,
fit as many items as possible into the trucks.

Data:

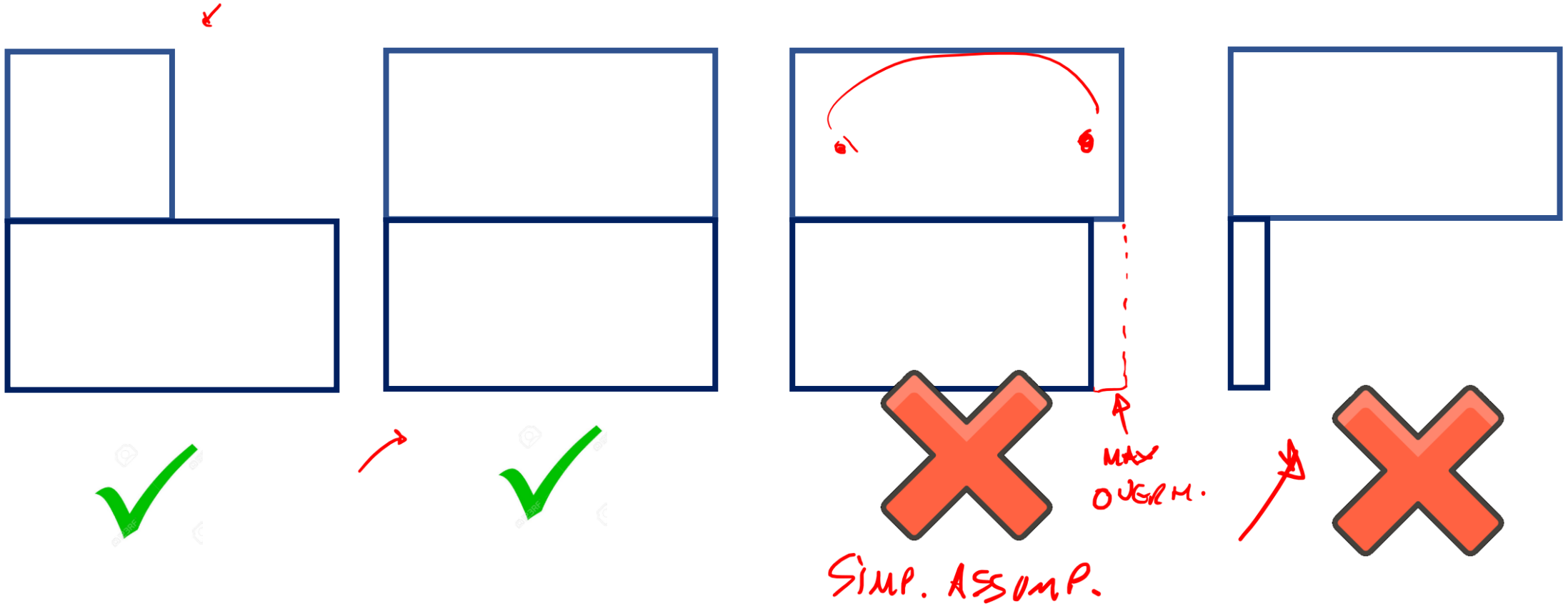
- Each item has
  - dimensions WxHxD
  - a weight
  - a due date
  - a value
- Each vehicle has
  - multiple compartments with known dimensions WxHxD
  - a maximum load (weight)
  - a departure date

Problem statement

Constraints

- The vehicle's weight capacity constraint must be observed
- Items cannot be placed on trucks departing after their due date
- The items must be arranged so that they fit into the truck
- They may be rotated the around the vertical axis, but can't be placed on their side
- Items may be placed on top of each other
  - Some may be marked "*top-load only*"
    - They must be supported from below with *no overhang*

No overhang



(Side view)

Problem statement

What we don't look at:

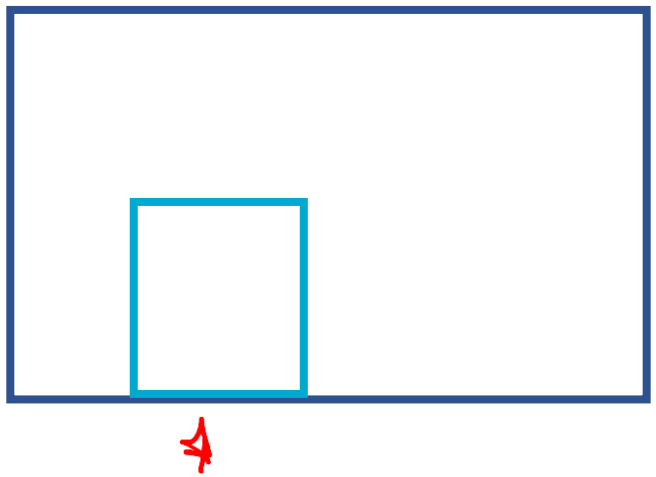
- Weight distribution / axle loading constraints
- Stacking constraints (e.g. item above cannot weigh more than item below)
- Tipping constraints (don't track centre of gravity)

Solution

- We will use a *metaheuristic* to solve the problem
- It will need two related subroutines:
  - Given a stack of items and a number of trucks, how can I distribute the items amongst the trucks
  - Given a truck and a stack of items, can I fit the items into the truck?
- We know that the metaheuristic will call these subroutines many thousands of times.
  - So, packing must be **FAST**

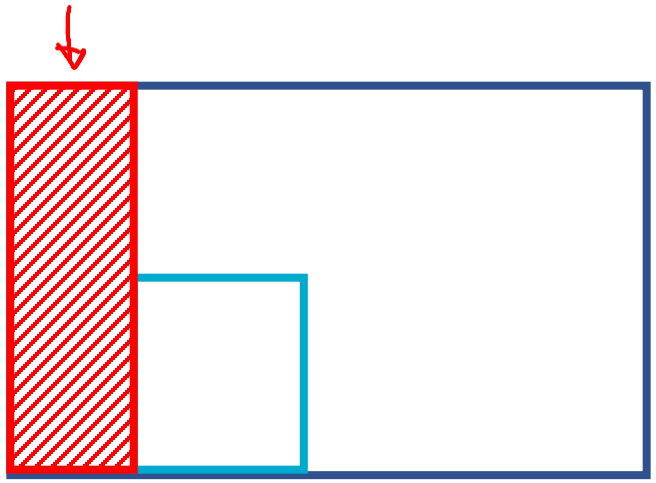
Solution

- The solution is based on a few observations.
- Observation 1: Might as well pack in corners



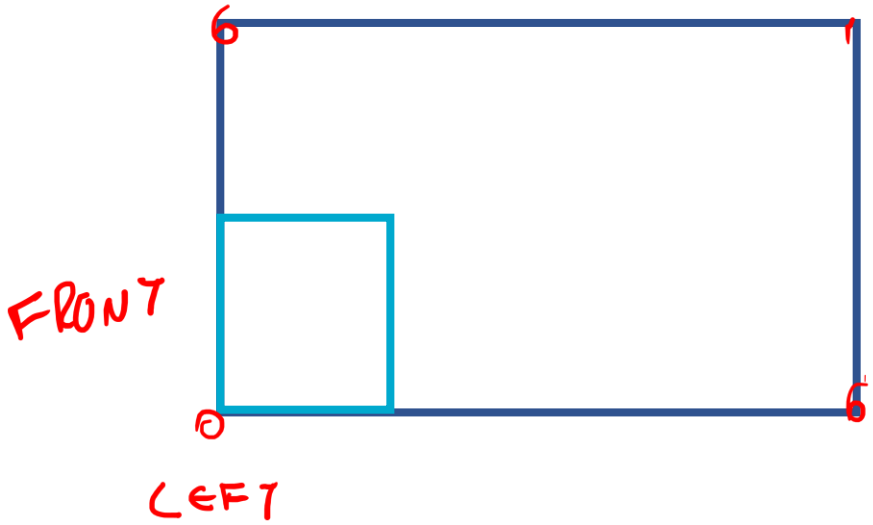
Solution

- The solution is based on a few observations.
- Observation 1: Might as well pack in corners



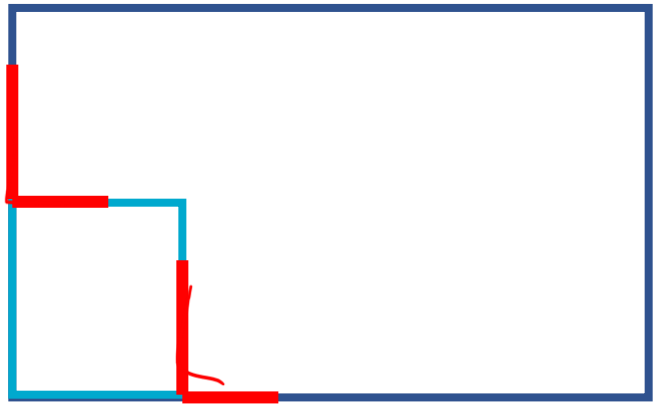
Solution

- The solution is based on a few observations.
- Observation 1: Might as well pack in corners



Solution

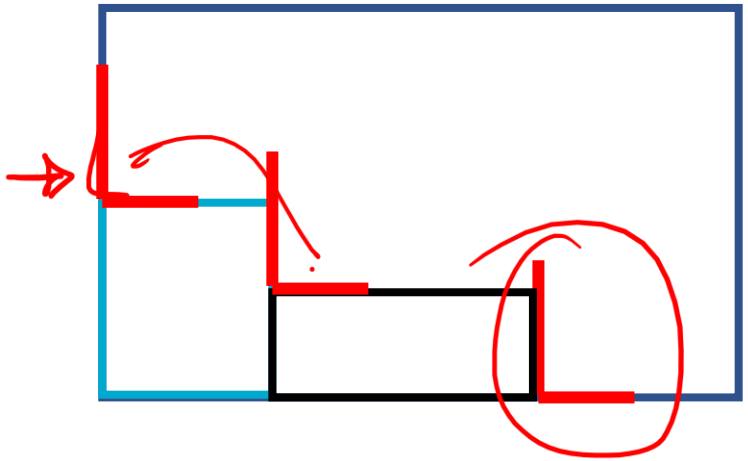
- The solution is based on a few observations.
- Observation 2: Corners are easy to track



Solution

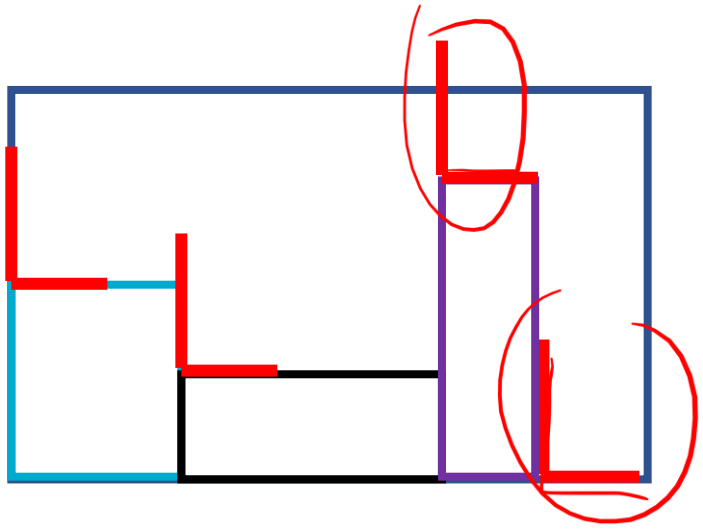
- The solution is based on a few observations.
- Observation 2: Corners are easy to track

Side view



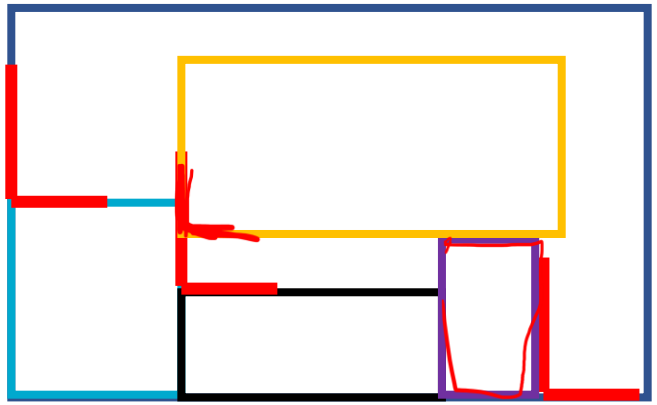
Solution

- The solution is based on a few observations.
- Observation 2: Corners are easy to track



Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track

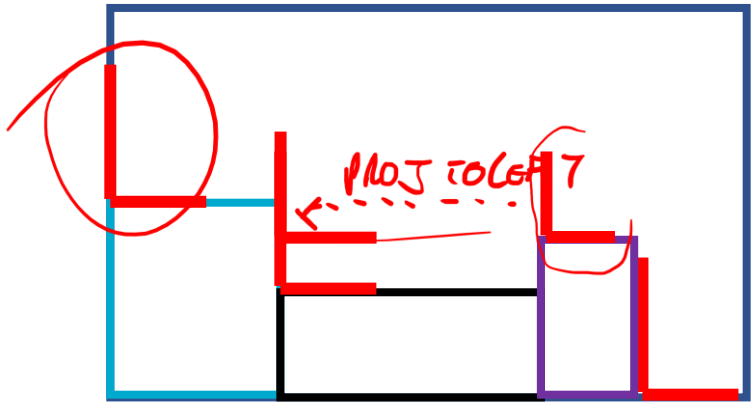


Recall that this is the view from the top:

- **Yellow is by the side of purple** (not on top)

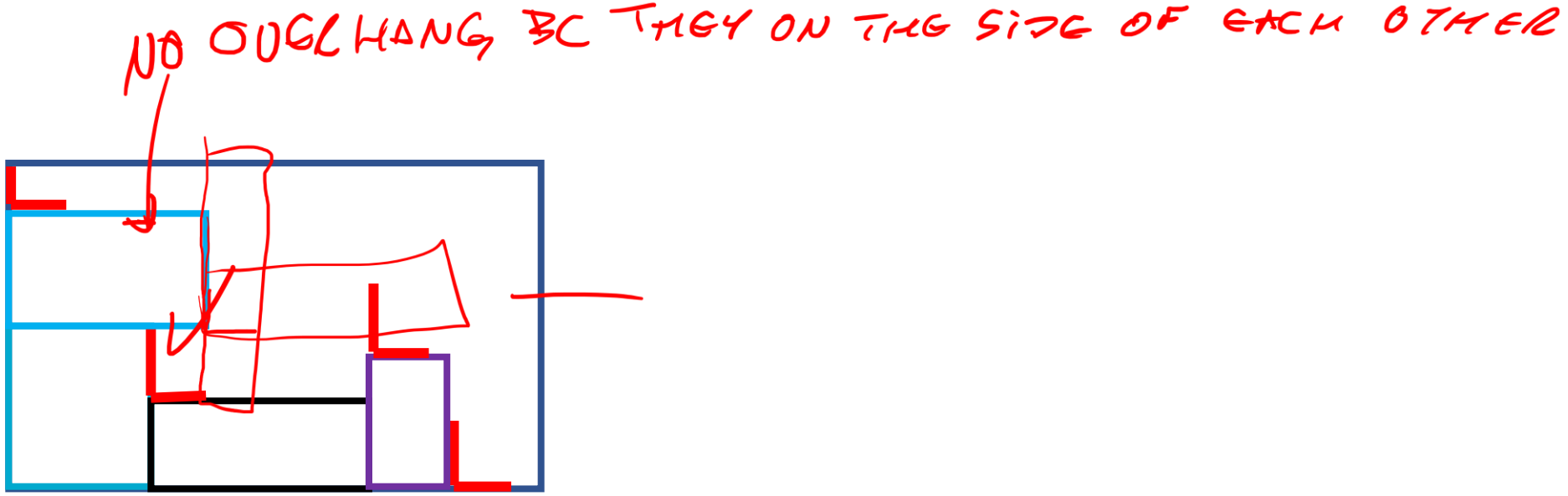
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



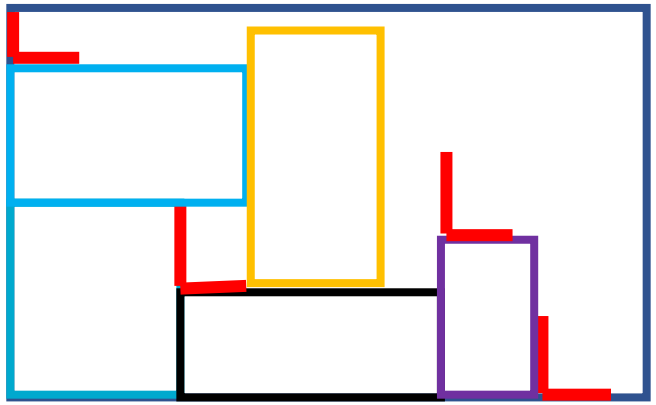
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



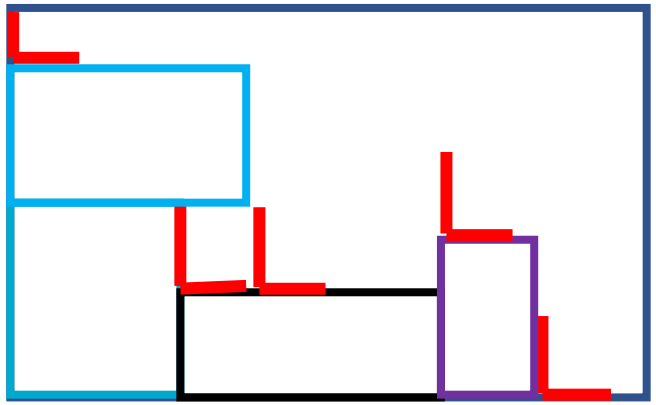
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



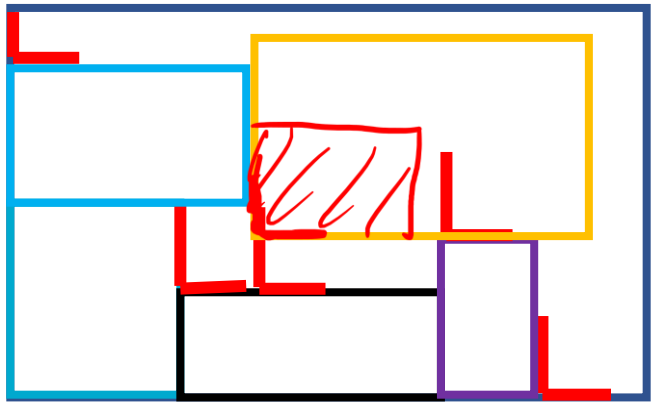
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



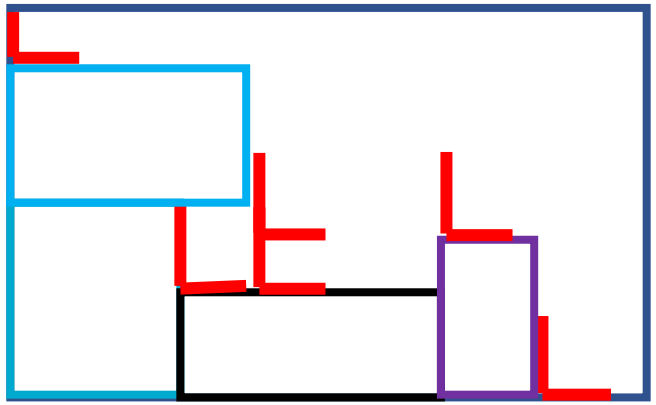
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



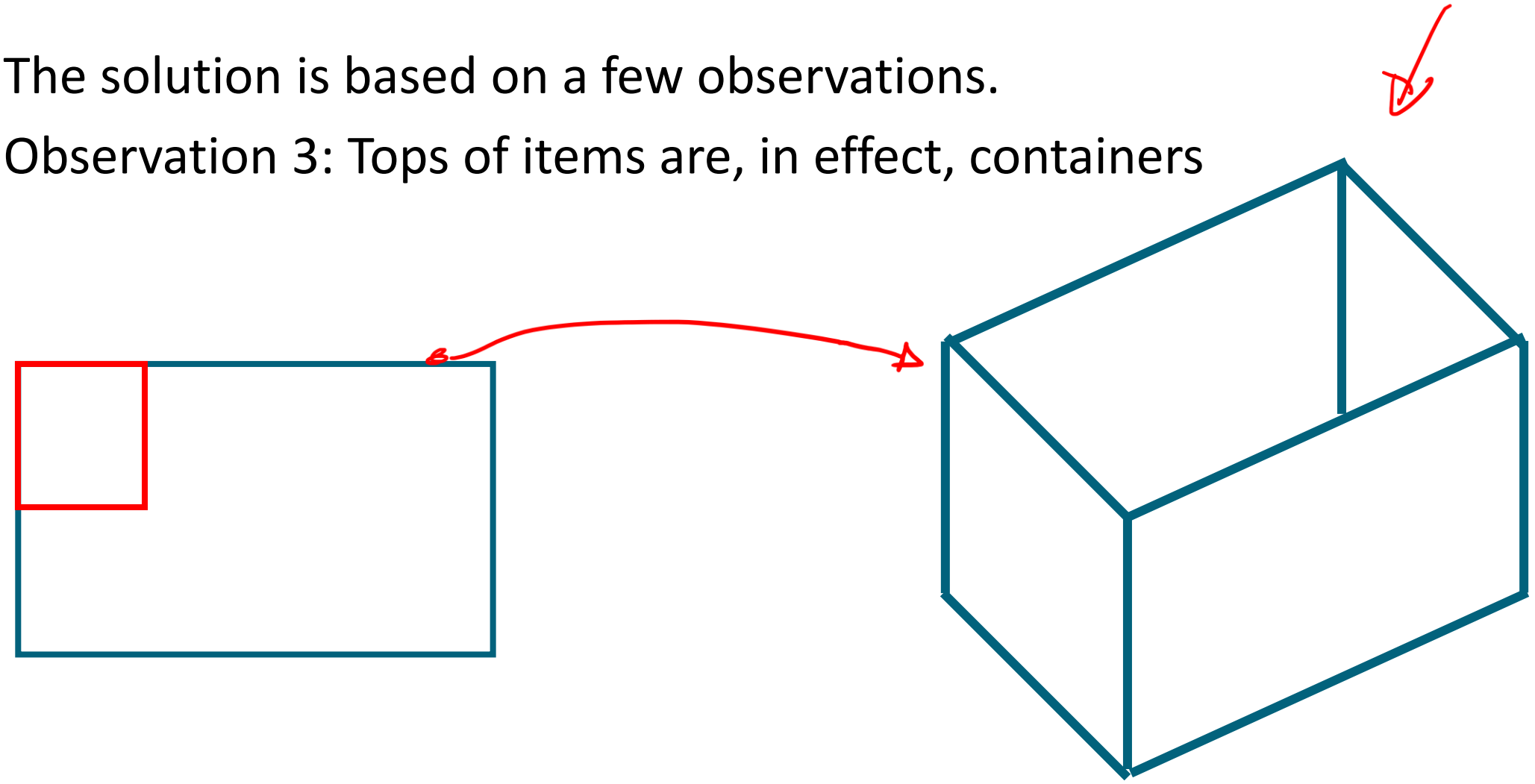
Solution

- The solution is based on a few observations.
- Observation 2’: Corners are mostly easy to track



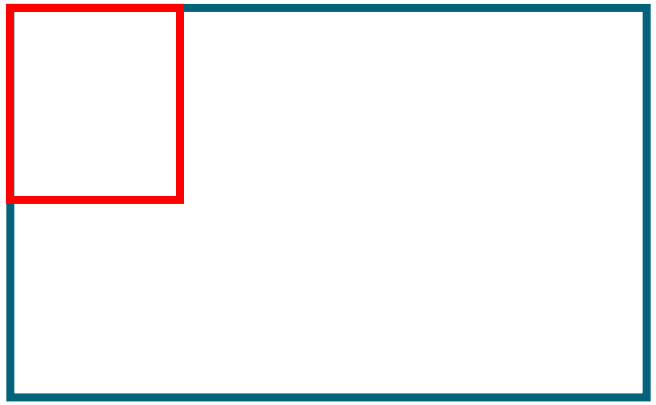
Solution

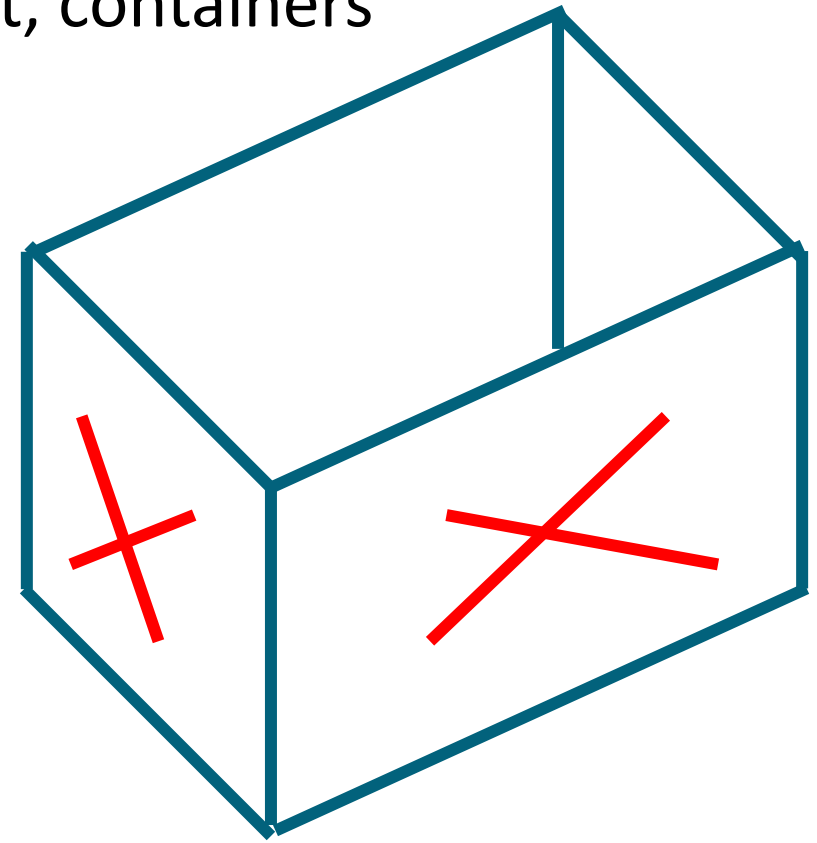
- The solution is based on a few observations.
- Observation 3: Tops of items are, in effect, containers



Solution

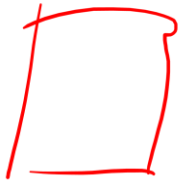
- The solution is based on a few observations.
- Observation 3: Tops of items are, in effect, containers

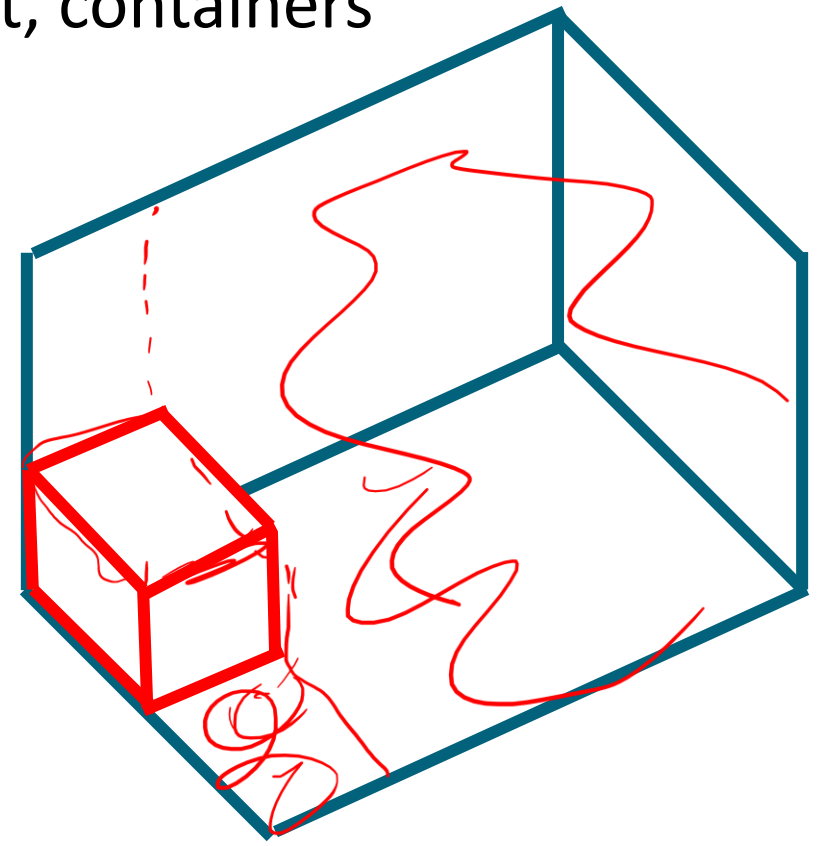




Solution

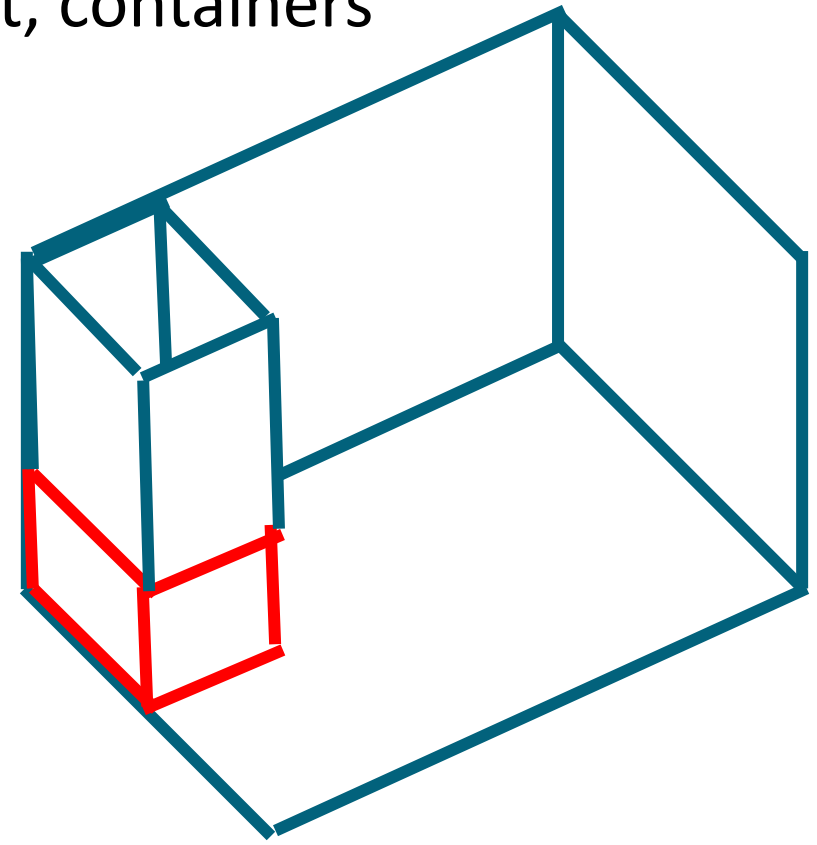
- The solution is based on a few observations.
- Observation 3: Tops of items are, in effect, containers





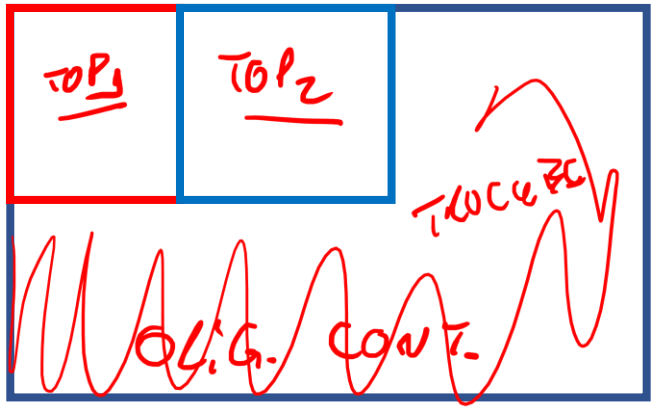
Solution

- The solution is based on a few observations.
- Observation 3: Tops of items are, in effect, containers



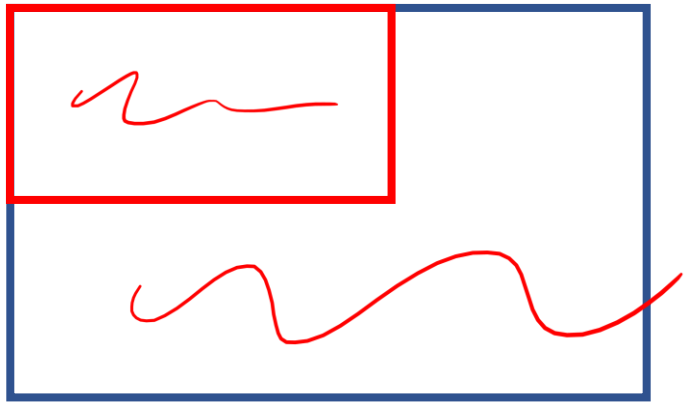
Solution

- The solution is based on a few observations.
- Observation 4: Adjoining tops (at the same height) can be joined



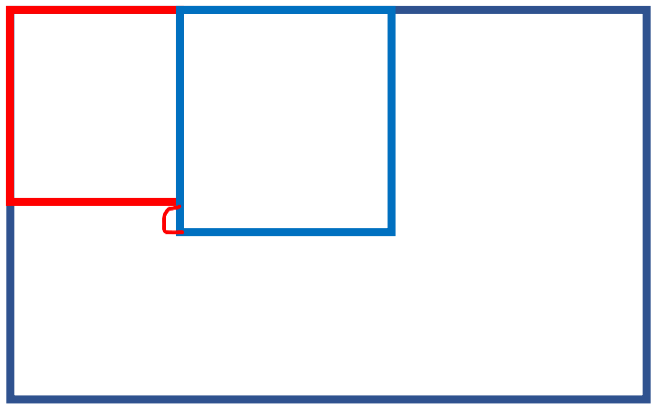
Solution

- The solution is based on a few observations.
- Observation 4: Adjoining tops (at the same height) can be joined



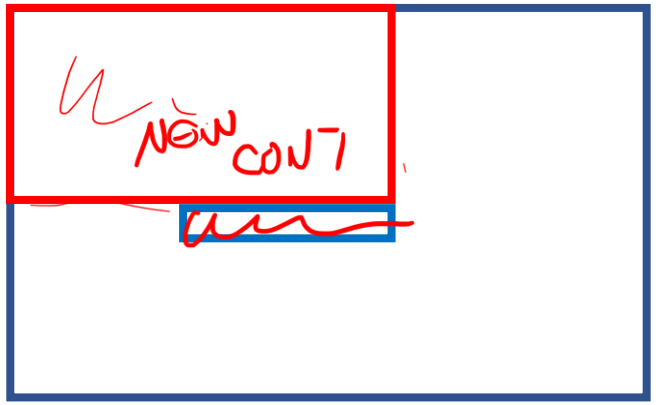
Solution

- The solution is based on a few observations.
- Observation 4: Adjoining tops (at the same height) can be joined



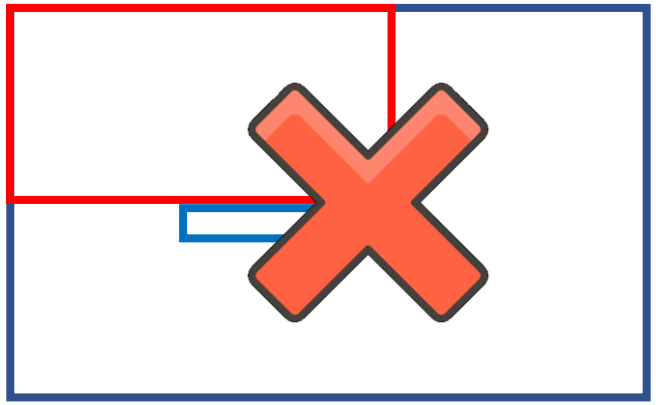
Solution

- The solution is based on a few observations.
- Observation 4: Adjoining tops (at the same height) can be joined



Solution

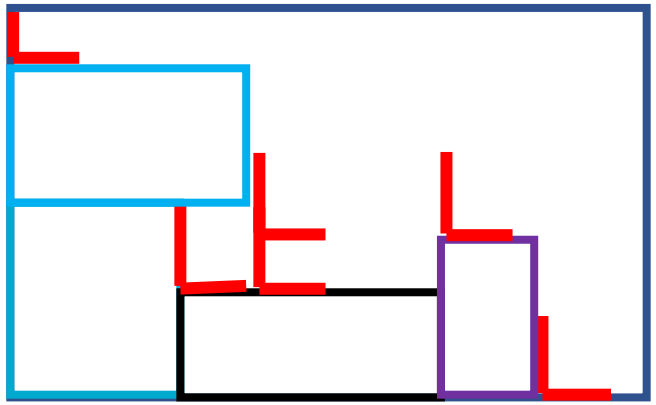
- The solution is based on a few observations.
- Observation 4: Adjoining tops (at the same height) can be joined



Solution

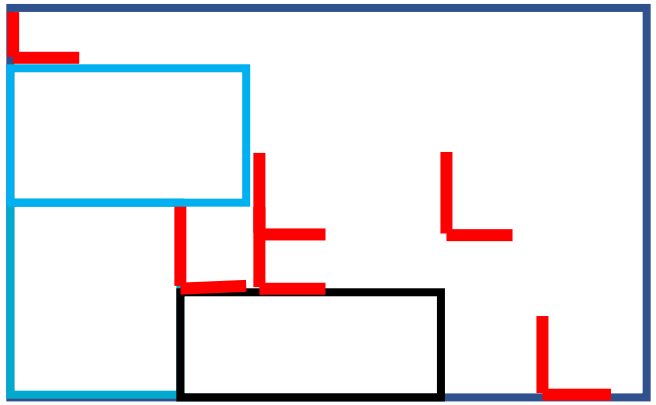
- The solution is based on a few observations.
- Observation 5: Taking things out of the middle is hard

EMPTY THE CAN?



Solution

- The solution is based on a few observations.
- Observation 5: Taking things out of the middle is hard

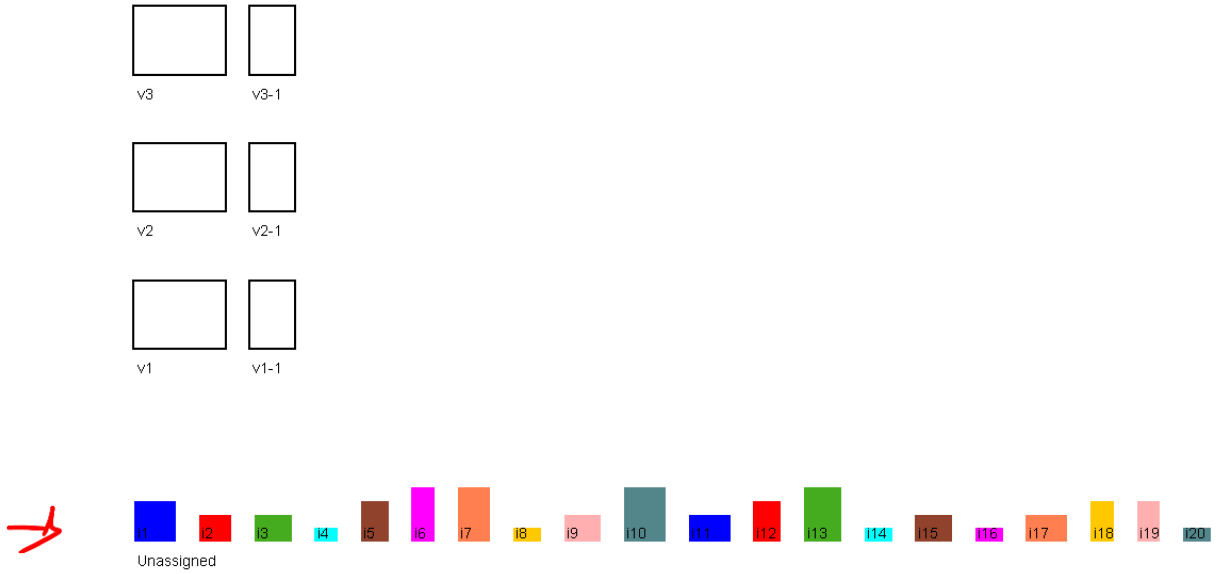


Solution

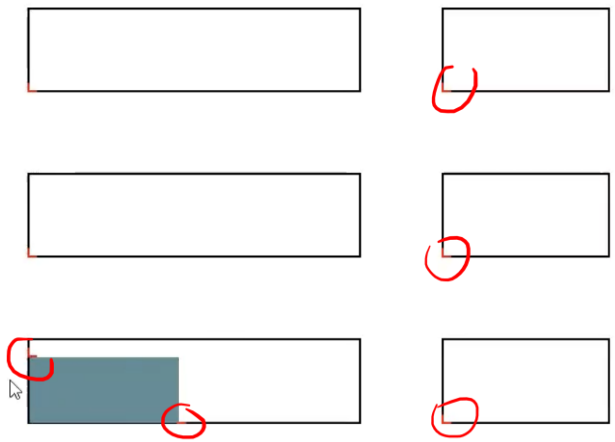
- The solution is based on a few observations.
- Observation 5: Taking things out of the middle is hard, so we just empty the container

Solution

- Let’s see it run …

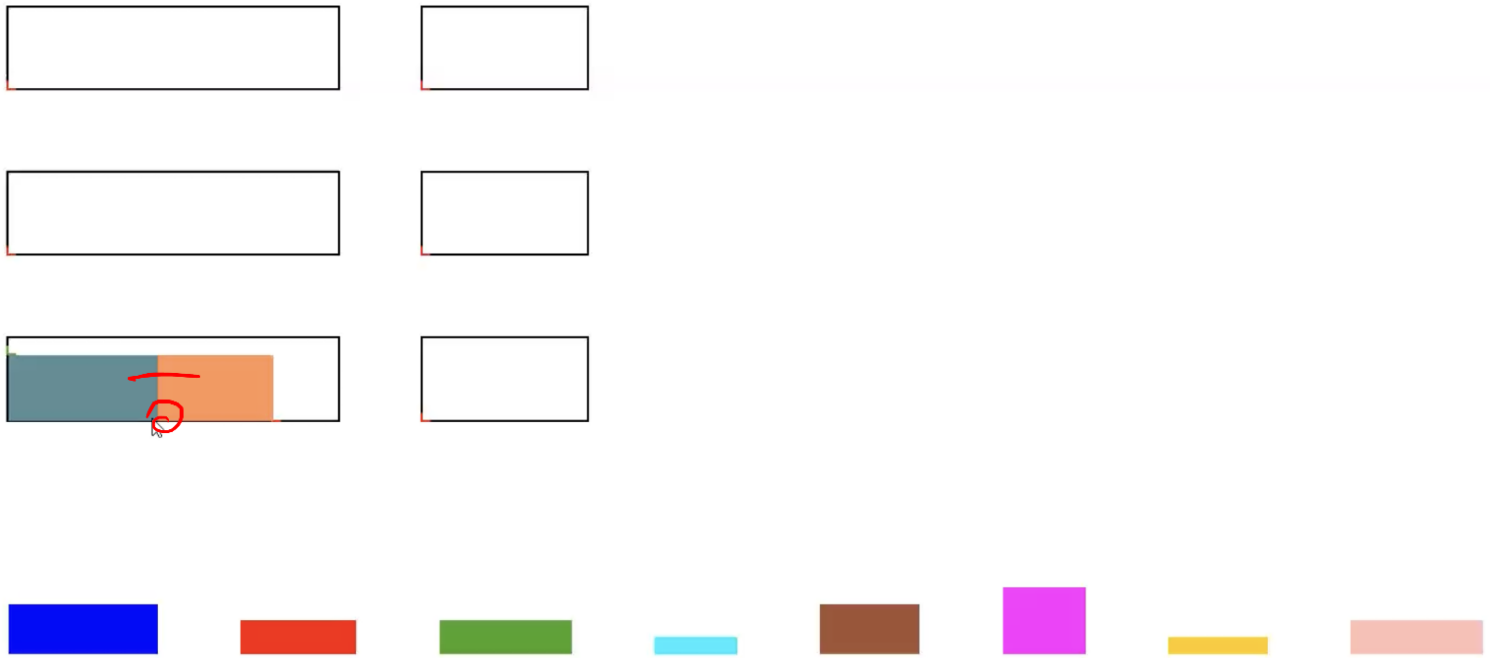


Solution



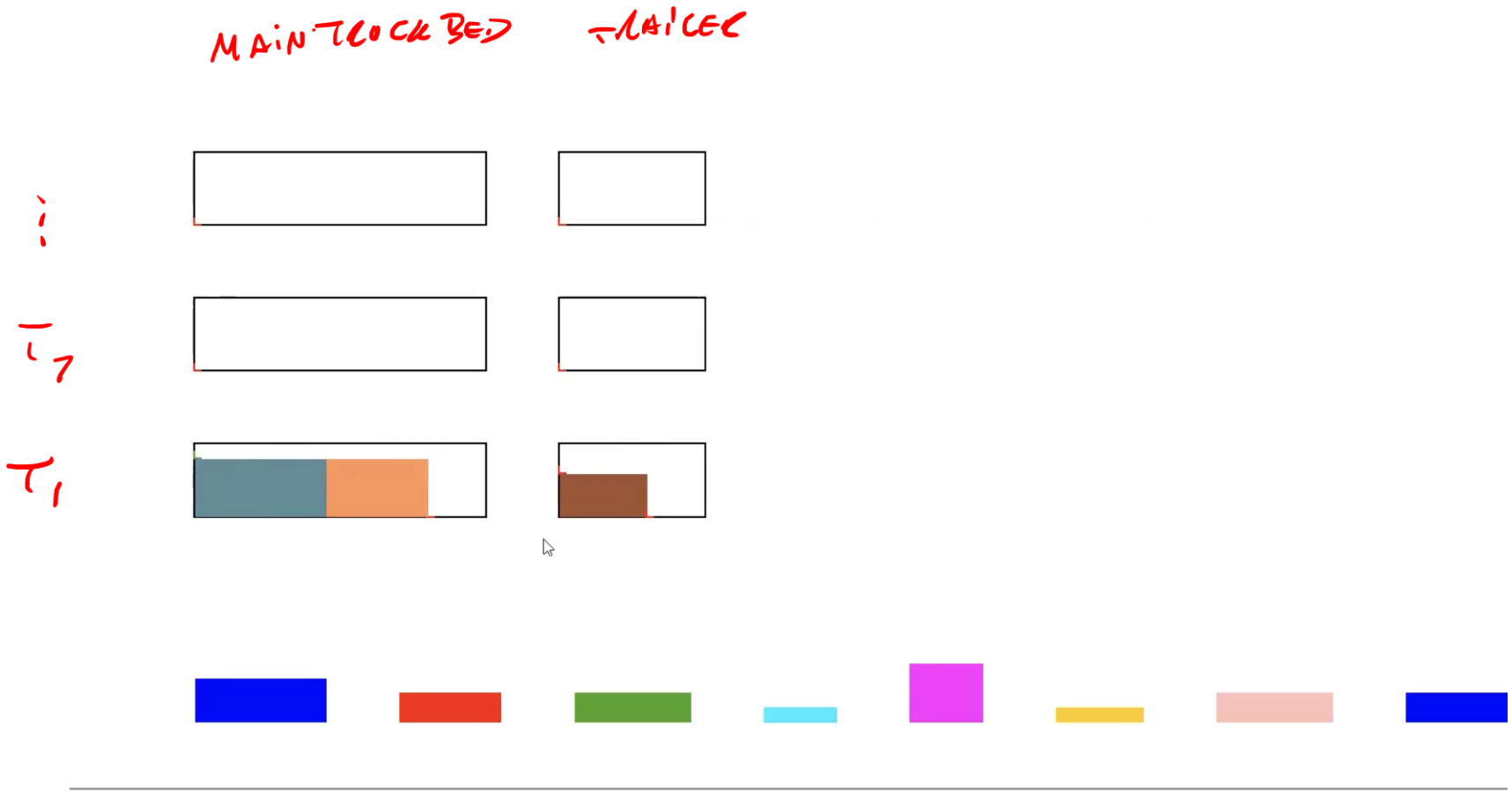
Solution

Solution

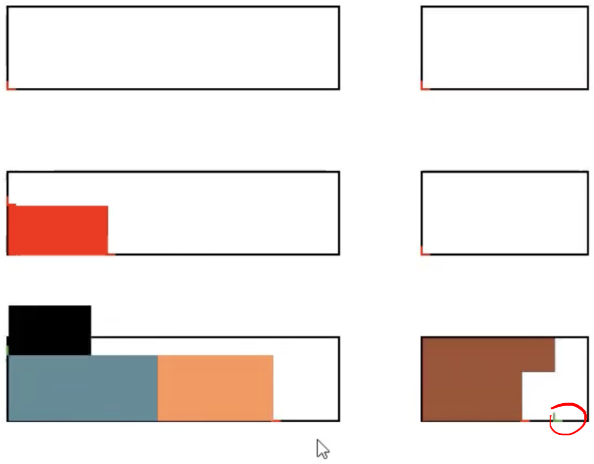


Solution

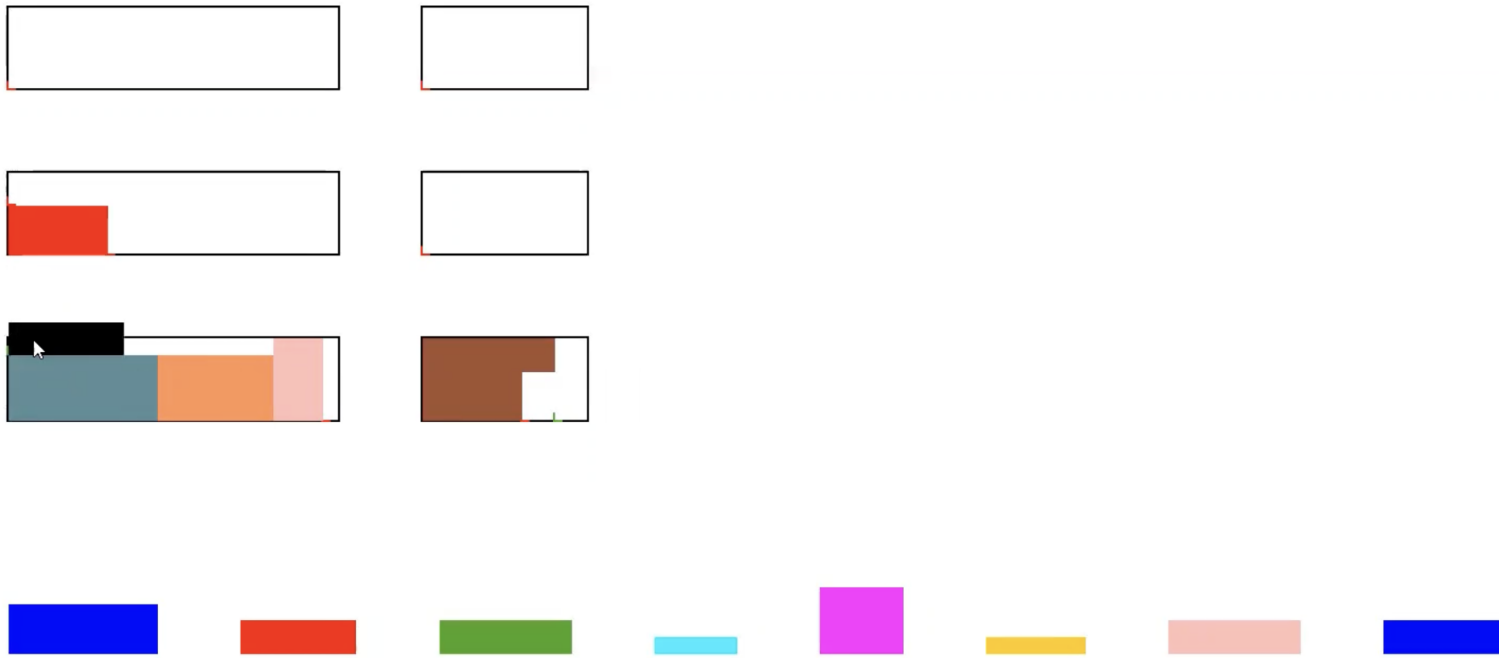
Solution



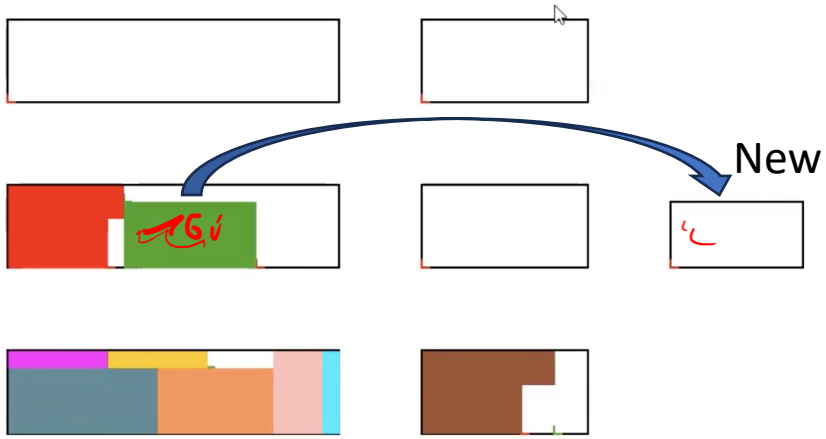
Solution



Solution

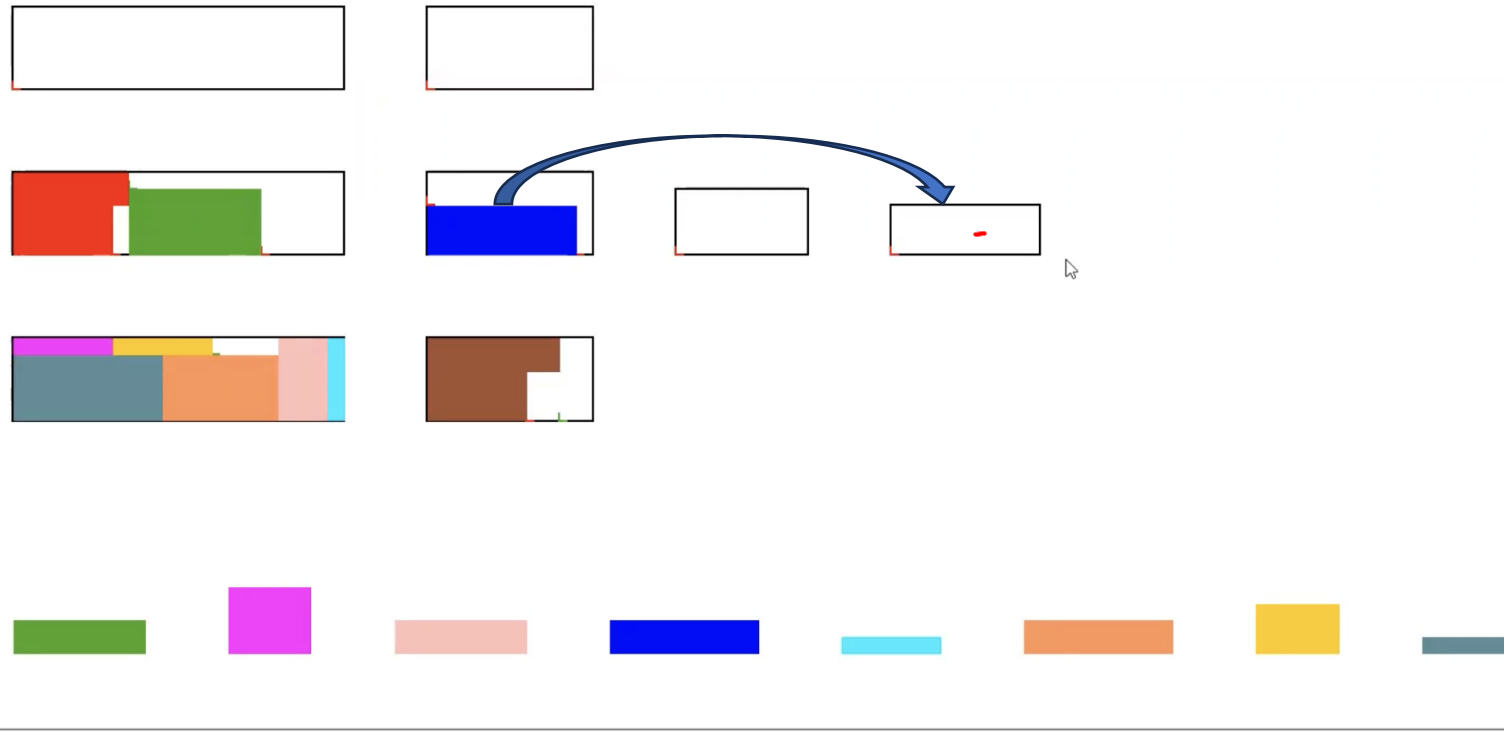


Solution

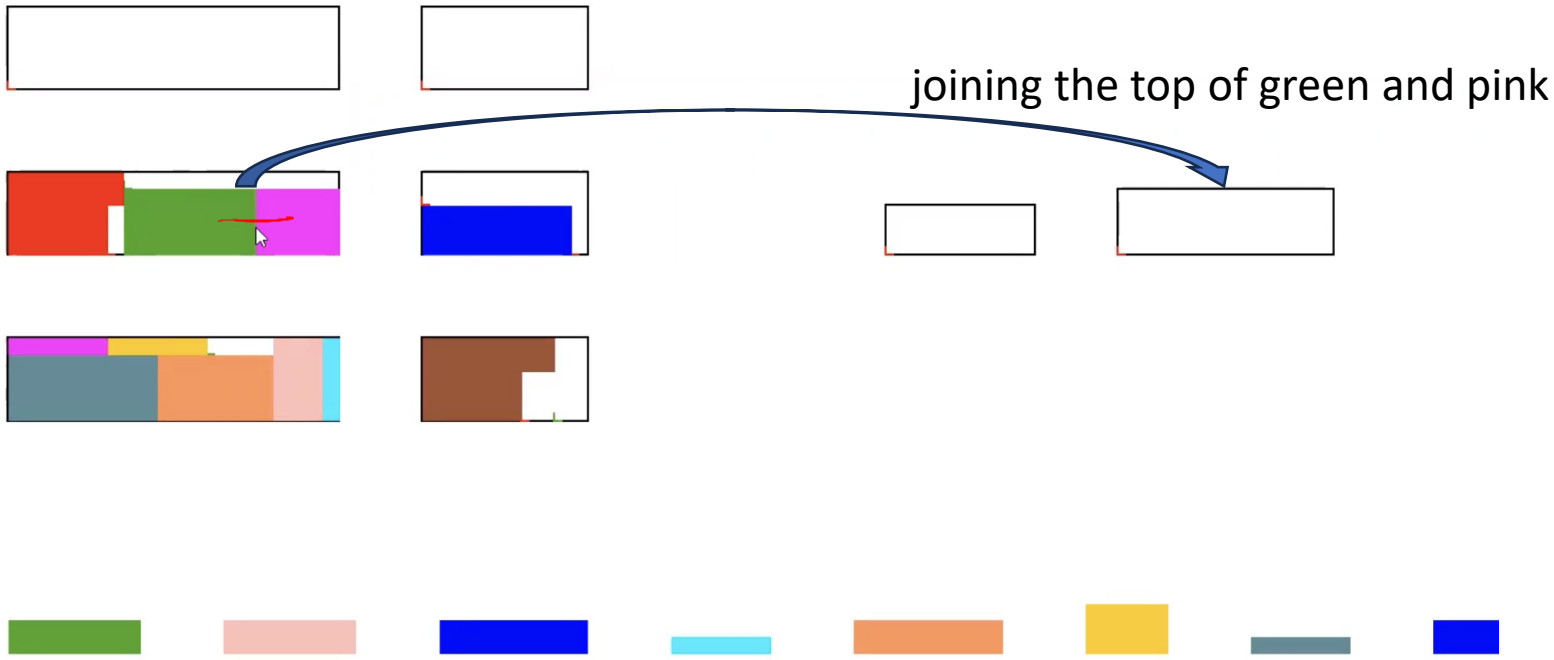


New container representing the top of green

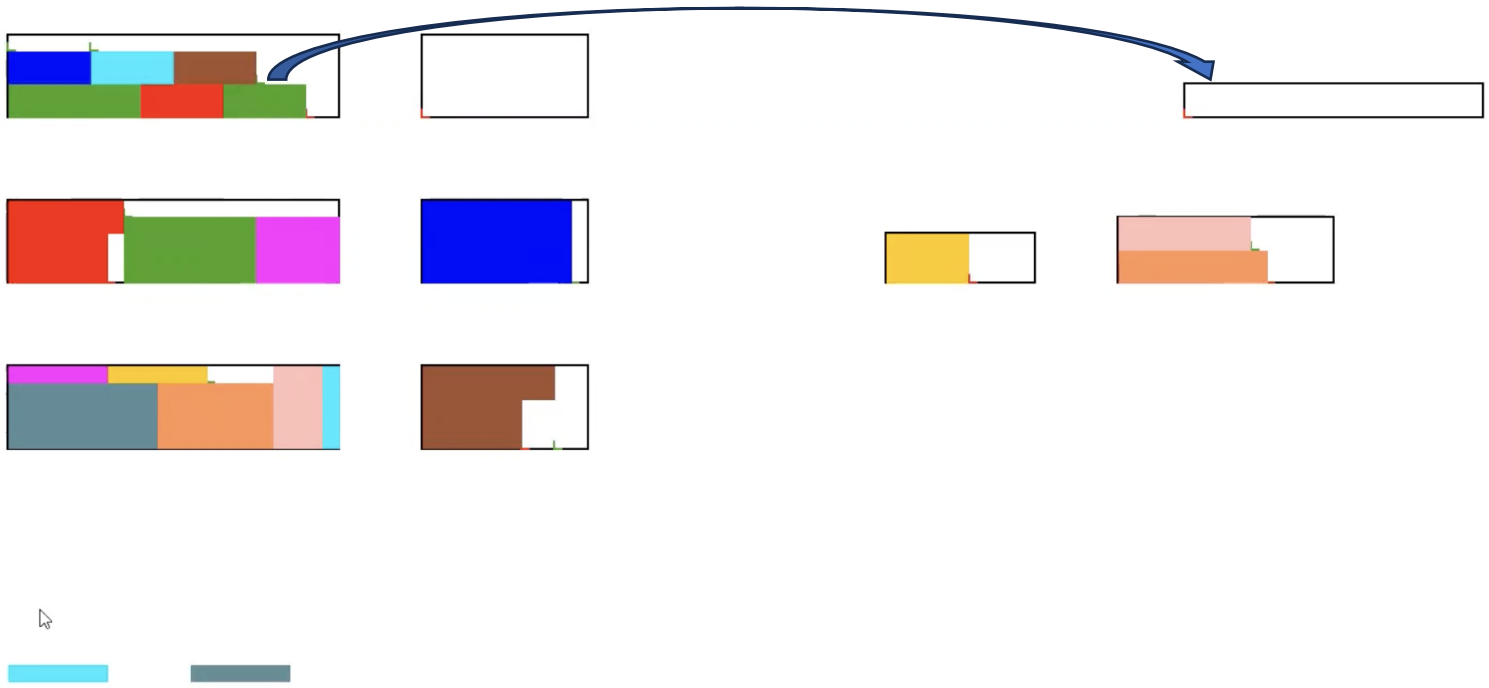
Solution



Solution



Solution



Conclusion

- Constructing a solution is the first step in solving a combinatorial optimisation problem
- Building up a solution one element at a time is often a good place to start
- Greedy heuristics can give half-good solutions, but are often structurally unsound
- Some techniques can give a heuristic a bit of foresight, such as Regret

LOOK AHEAD

Next time

- Now we have our first solution … can we improve it?

FRIDAY QUIZ