The Vehicle Routing Problem with Time Windows

Dr Philip Kilby | Team Leader, Optimisation Applications and Platforms

June 2017

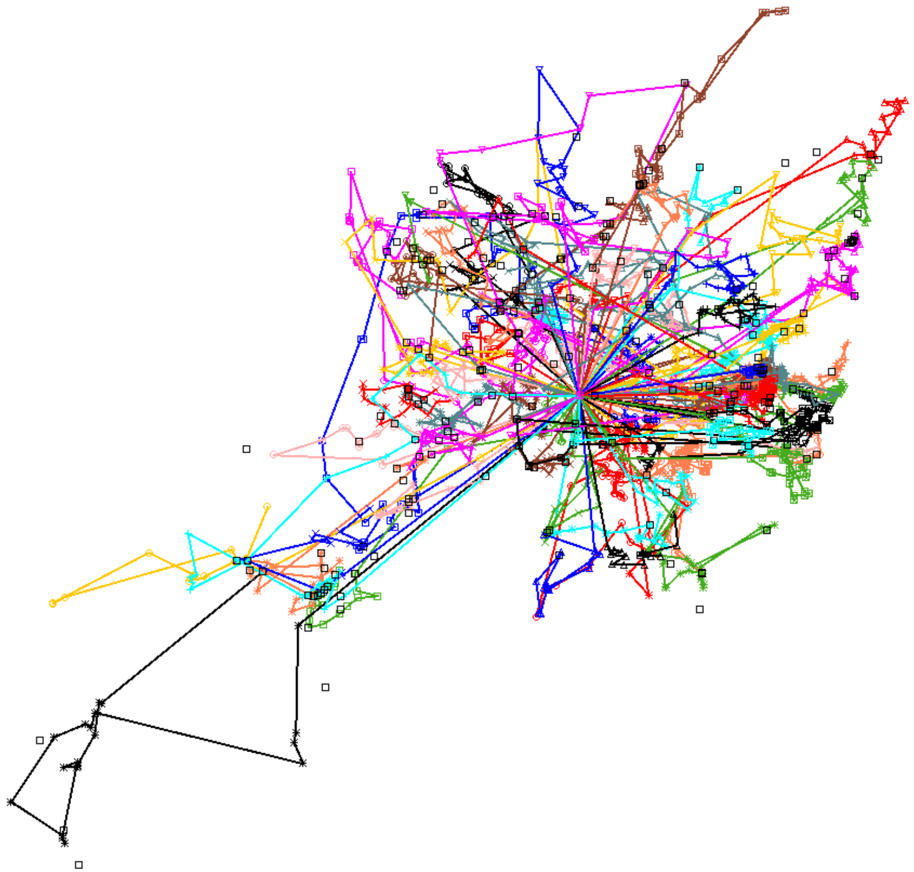
www.data61.csiro.au

Outline

- Problem Description
- Solving the VRP
  - Construction
  - Local Search
  - Meta-heuristics
    - Variable Neighbourhood Search
    - Including Large Neighbourhood Search
- CP101
- A CP model for the VRP

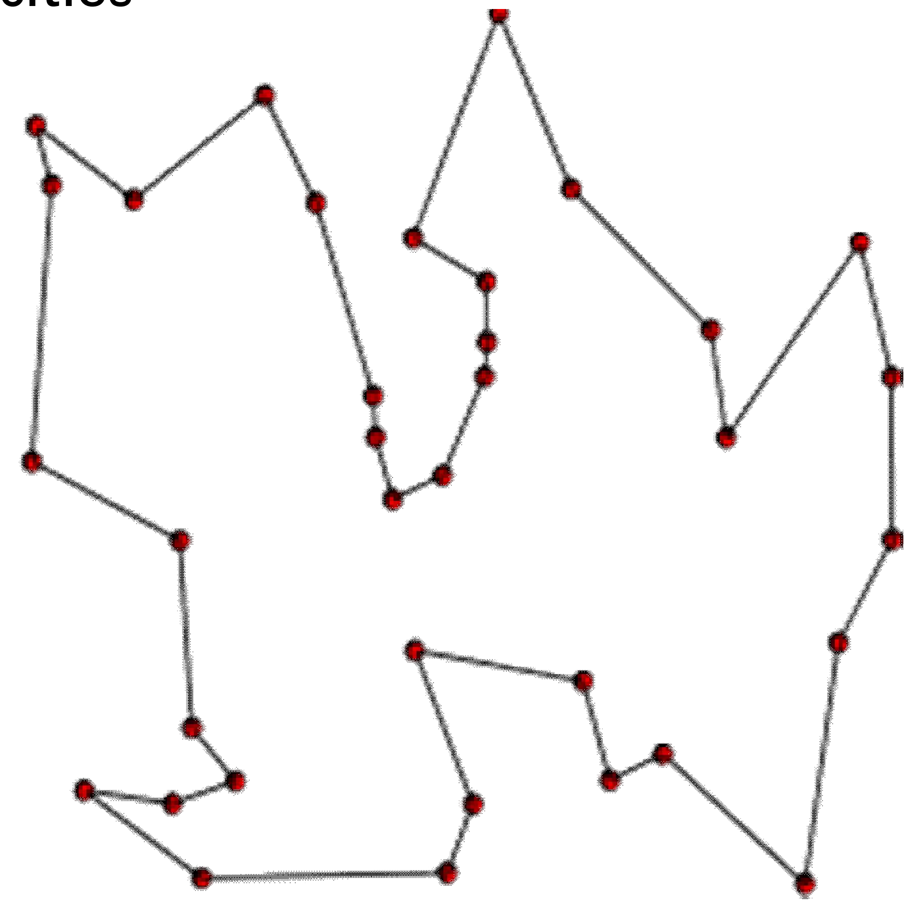
Vehicle routing problem

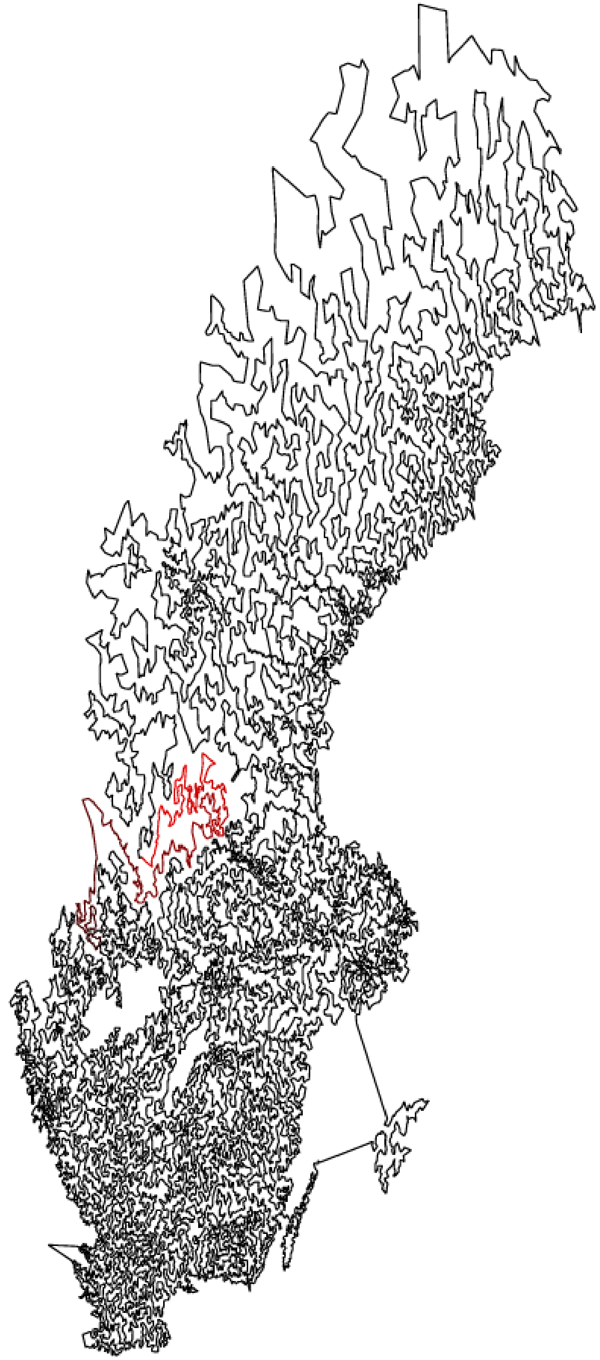
Given a set of customers, and a fleet of vehicles to make deliveries, find a set of routes that services all customers at minimum cost



Travelling Salesman Problem

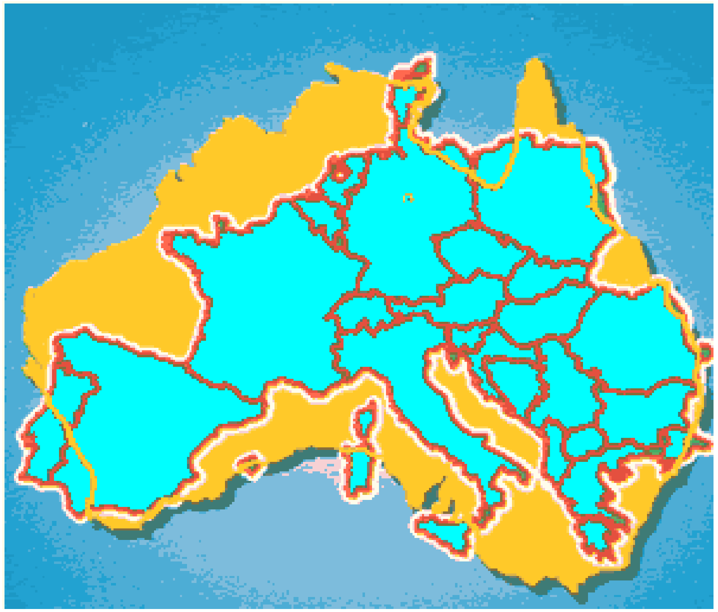
- Find the tour of minimum cost that visits all cities

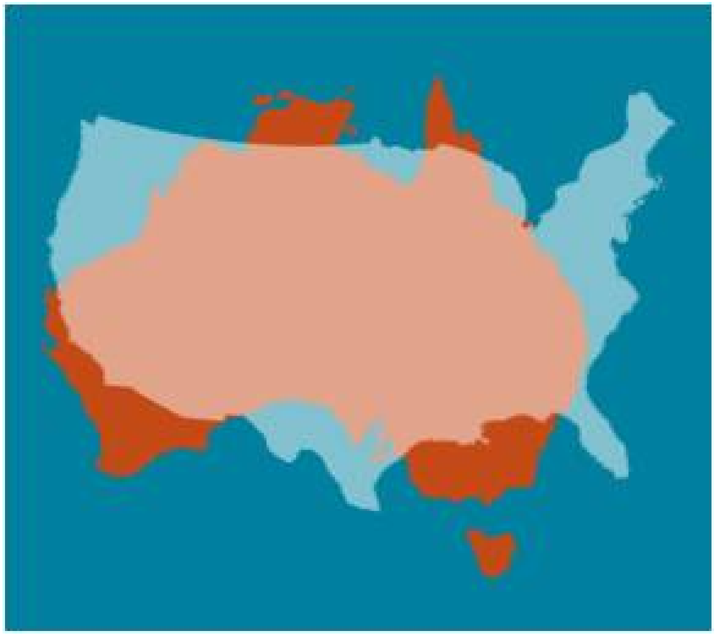




Why study the VRP?

- It’s hard: it exhibits all the difficulties of comb. opt.
- It’s useful:
  - The logistics task is 9% of economic activity in Australia
  - Logistics accounts for 10% of the selling price of goods





Why study the VRP in Robotics?

Appears as a sub-problem

Task allocation to agents

- Multiple agents with multiple tasks
- Best allocation minimizes cost

Scheduling with setup costs

- Can be modelled as a VRPTW

Vehicle Routing Problem

For each customer, we know

- Quantity required
- The cost to travel to every other customer

For the vehicle fleet, we know

- The number of vehicles
- The capacity

We must determine which customers each vehicle serves, and in what order, to minimise cost

Vehicle Routing Problem

Objective function

- In academic studies, usually a combination:
    - o First, minimise number of routes
    - o Then minimise total distance or total time

- In real world
    - o A combination of time and distance
    - o Must include vehicle- and staff-dependent costs
    - o Usually vehicle numbers are fixed
    - o Includes “preferences” – like pretty routes

Time window constraints

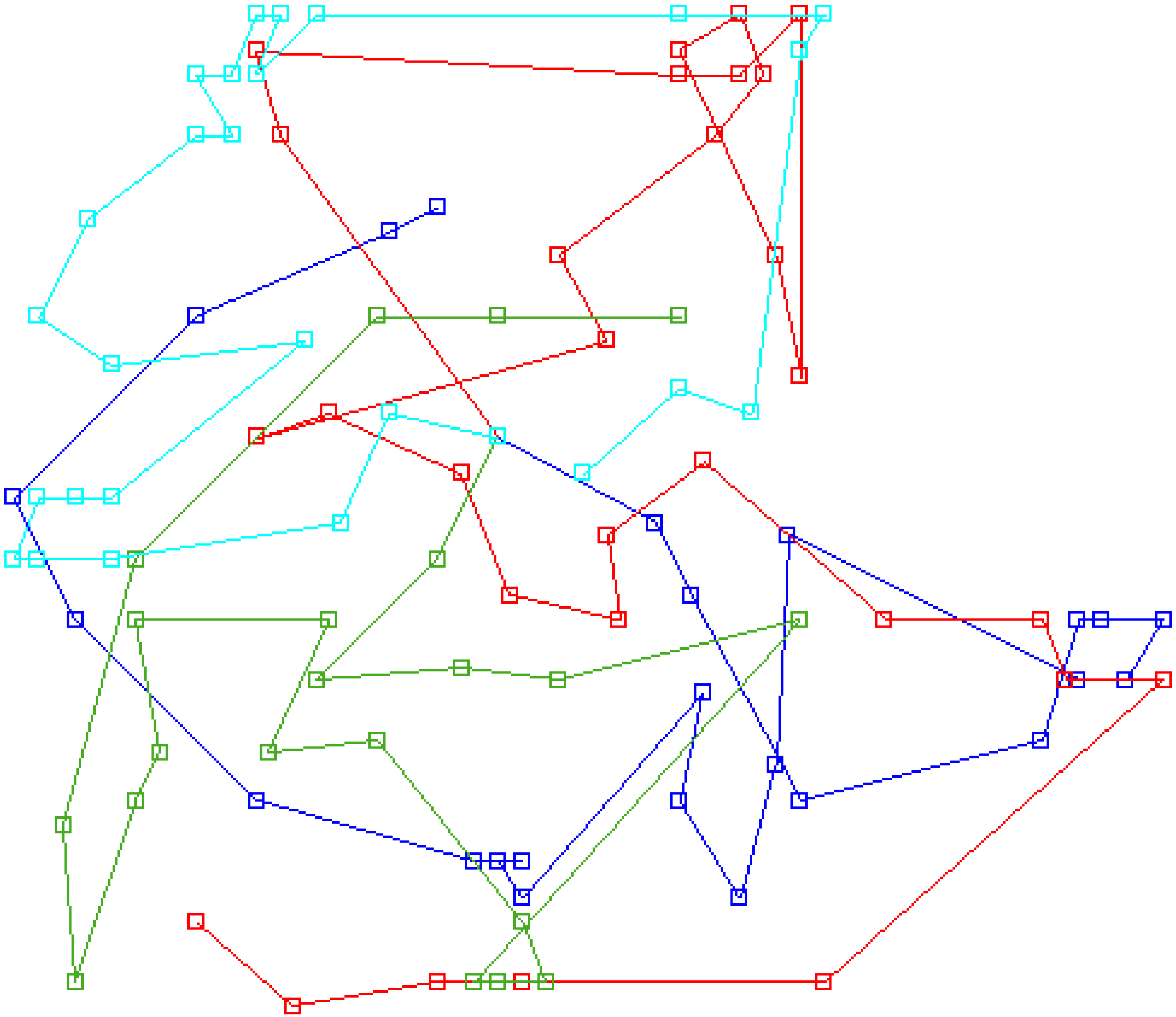
Vehicle routing with Constraints

- Time Window constraints
  - A window during which service can start
  - E.g. only accept delivery 7:30am to 11:00am

  - Additional input data required
    - Duration of each customer visit
    - Time between *each pair* of customers

    - (Travel time can be vehicle-dependent or time-dependent)
  - Makes the route harder to visualise

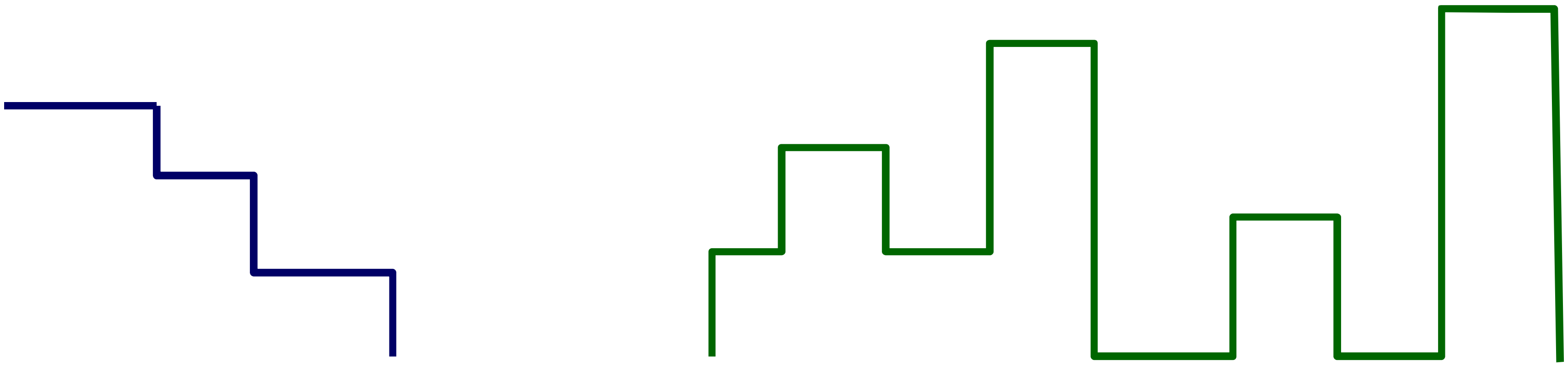
Time Window constraints



Pickup and Delivery problems

- Most routing considers delivery to/from a depot (depots)
- Pickup and Delivery problems consider FedEx style problem:
  - *pickup at location A, deliver to location B*

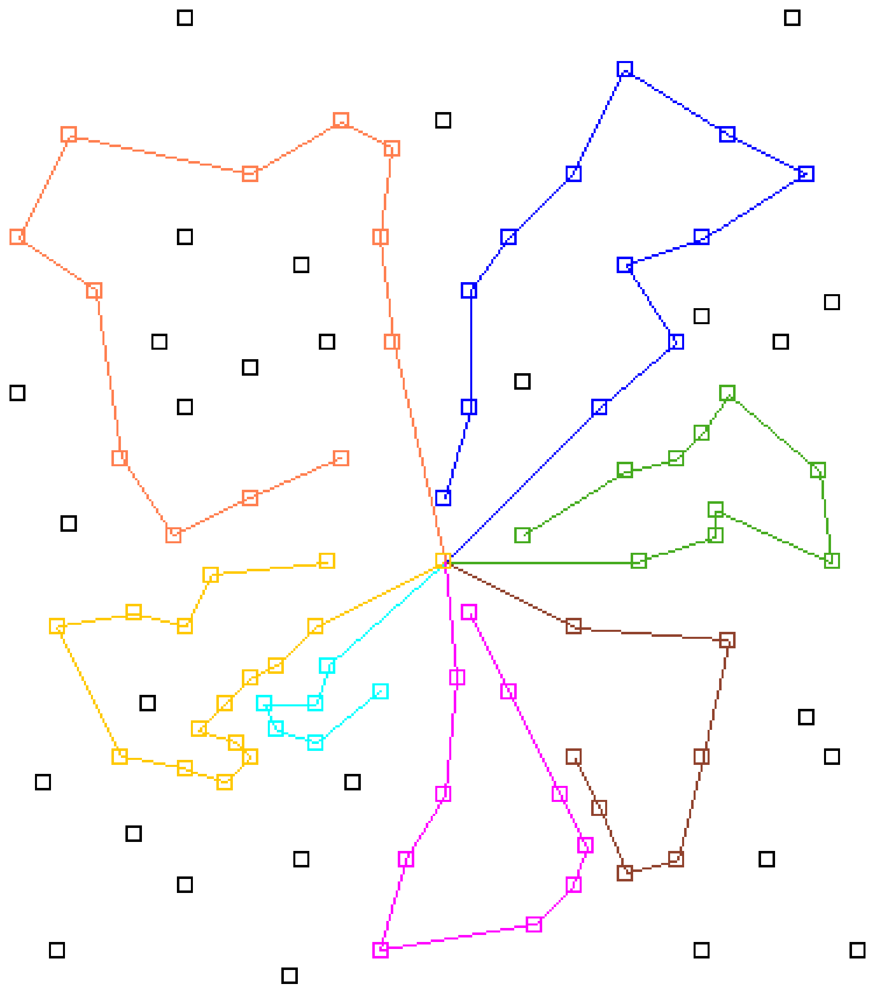
- Load profile:



Other variants

Profitable tour problem

- Not all visits need to be completed
- Known profit for each visit
- Choose a subset that gives maximum *profit* = (*revenue from visits*) – (*routing cost*)



Orienteering Problem

- Maximum revenue in limited time

VRP meets the real world

Many groups now looking at real-world constraints

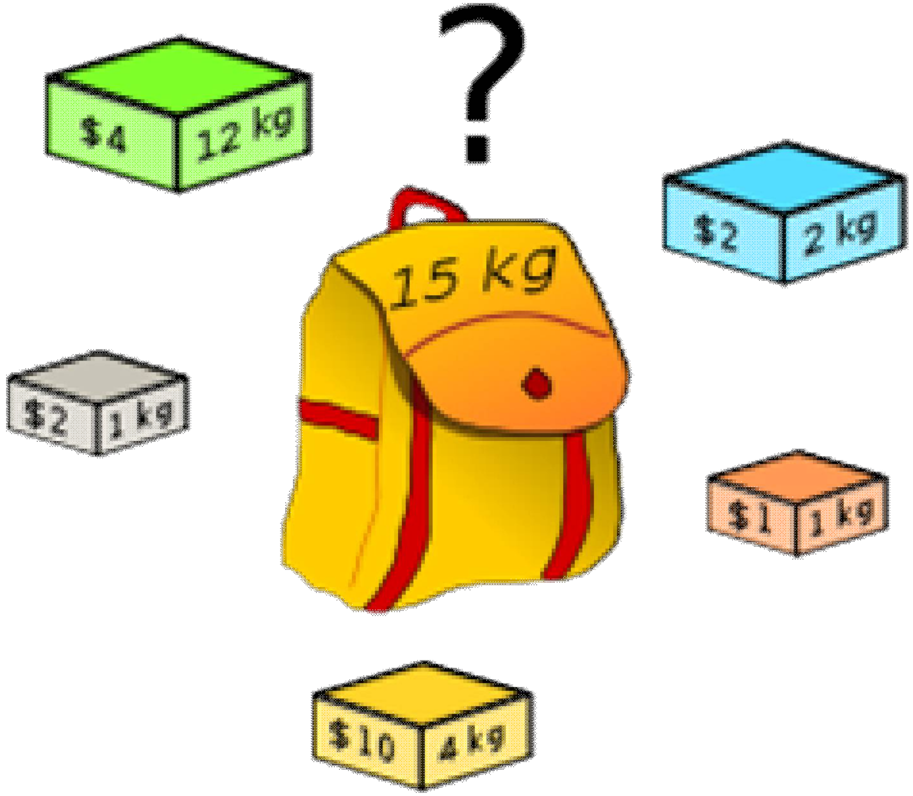
Rich Vehicle Routing Problem

- Attempt to model constraints common to many real-life enterprises
  - Multiple Time windows
  - Multiple Commodities
  - Multiple Depots
  - Heterogeneous vehicles
  - Compatibility constraints
    - Goods for customer *A* {must | can’t} travel with goods from customer *B*
    - Goods for customer *A* {must | can’t} travel on vehicle *C*

VRP as an instance

VRP is a Combinatorial Optimization problem

- Others include
  - o Scheduling
  - o Assignment
  - o Bin Packing



Solving VRPs

Solution Methods

Exact:

- Integer Programming or Mixed Integer Programming
- Constraint Programming

Heuristic:

- Construct
- Improve
  - Local Search
  - Meta-heuristics

Exact Methods

VRP:

- MIP: Can only solve problems with 100-150 customers
- CP: Similar size

ILP

$$\text{minimise}: \sum_{i,j} c_{ij} \sum_{k} x_{ijk}$$

x_{ijk} = 1 if vehicle *k* travels directly from *i* to *j*

subject to

$$\sum_{i} \sum_{k} x_{ijk} = 1 \quad \forall j$$

Exactly one vehicle in

$$\sum_{j} \sum_{k} x_{ijk} = 1 \quad \forall i$$

Exactly one vehicle out

$$\sum_{j} \sum_{k} x_{ihk} - \sum_{j} \sum_{k} x_{hjk} = 0 \quad \forall k, h$$

It's the same vehicle

$$\sum_{i} q_i \sum_{j} x_{ijk} \leq Q_k \quad \forall k$$

Capacity constraint

$$\sum x_{ijk} = |S| - 1 \quad S \subseteq \mathcal{P}(N), 0 \notin S$$

Subtour elimination

$$x_{ijk} \in \{0,1\}$$

ILP

$$\text{minimise}: \sum_{i,j} c_{ij} \sum_k x_{ijk}$$

subject to

$$\sum_i \sum_k x_{ijk} = 1 \quad \forall j$$

$$\sum_j \sum_k x_{ijk} = 1 \quad \forall i$$

$$\sum_j \sum_k x_{ihk} - \sum_j \sum_k x_{hjk} = 0 \quad \forall k, h$$

$$\sum_i q_i \sum_j x_{ijk} \leq Q_k \quad \forall k$$

$$\sum x_{ijk} = |S| - 1 \quad S \subseteq \mathcal{P}(N), 0 \notin S$$

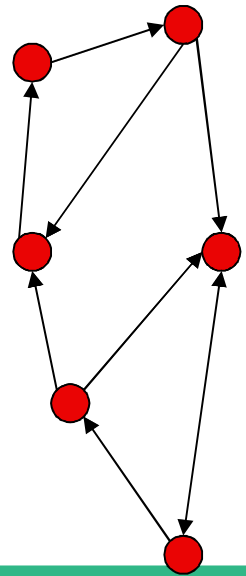
$$x_{ijk} \in \{0,1\}$$

Advantages

- Can find optimal solution

Disadvantages

- Only works for small problems
- One extra constraint → back to the drawing board
- *S* is huge



Depot

ILP – Column Generation

Columns represent routes

Column/route cost c_k

Rows represent customers

Array entry $a_{ik} = 1$ iff customer i is covered by route k

	89	*76*	*99*	*45*	*32*	
1	1	1	1	0	0	…
2	0	1	1	0	1	…
3	0	0	0	0	0	…
4	1	0	1	1	0	…
5	1	0	0	0	0	…

Column Generation

- Decision var x_k: Use column k?
- Column only appears if feasible ordering is possible
- Cost of best ordering is c_k
- Best order stored separately
- Master problem at right

$$\min \quad \sum c_k x_k$$

subject to

$$\lambda_i \longrightarrow \quad \sum_k a_{ik} x_k = 1 \quad \forall i$$

$$x_k \in \{0,1\}$$

Heuristics for the VRP

Heuristics:

Often variants of

- Construct
- Improve

Heuristics for the VRP

Construction by Insertion

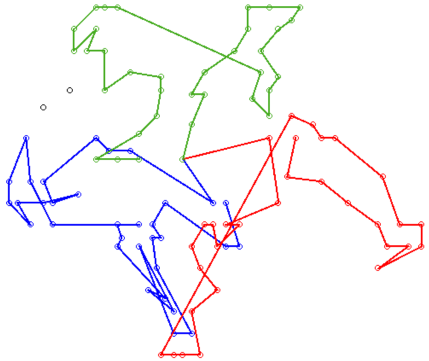
- Start with an empty solution
- Repeat
  - Choose which customer to insert
  - Choose where to insert it

E.g. (Greedy)

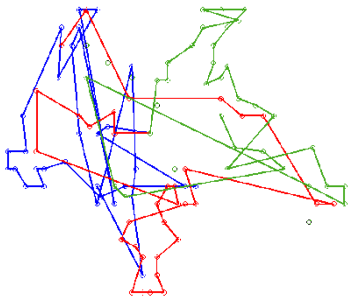
- Choose the customer that increases the cost by the least
- Insert it in the position that increases the cost by the least

Solving the VRP the easy way

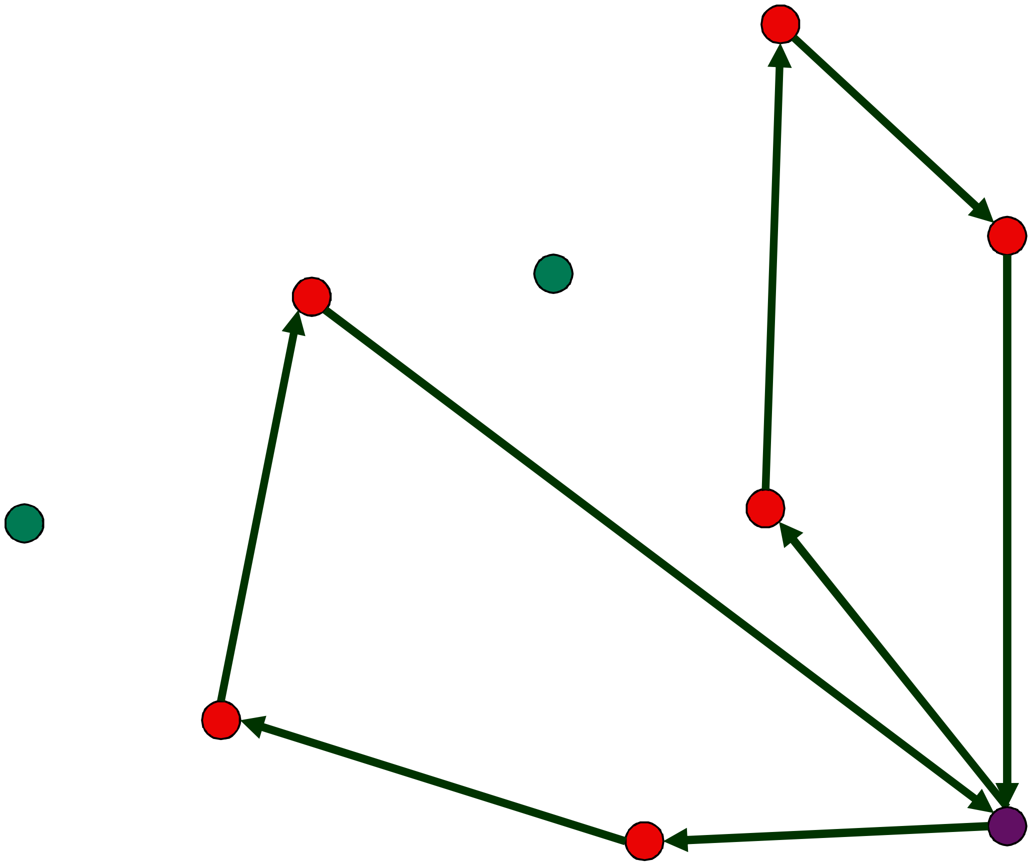
Insert methods



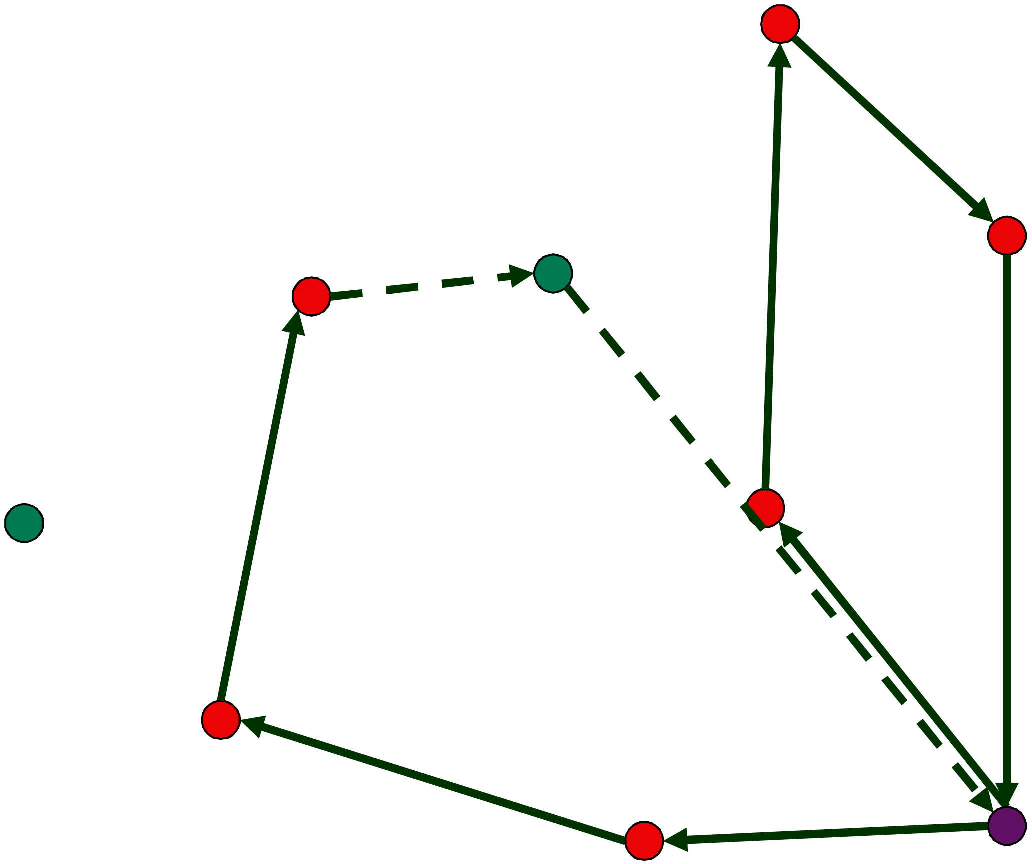
Order is important:



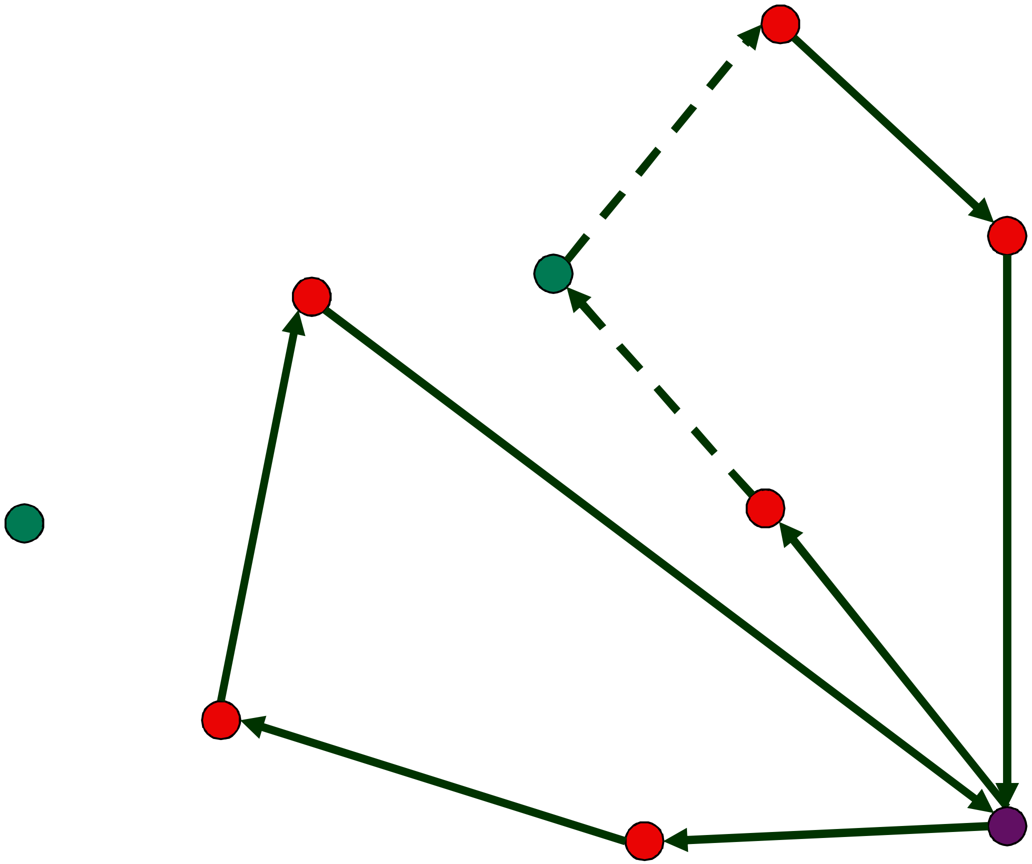
Regret



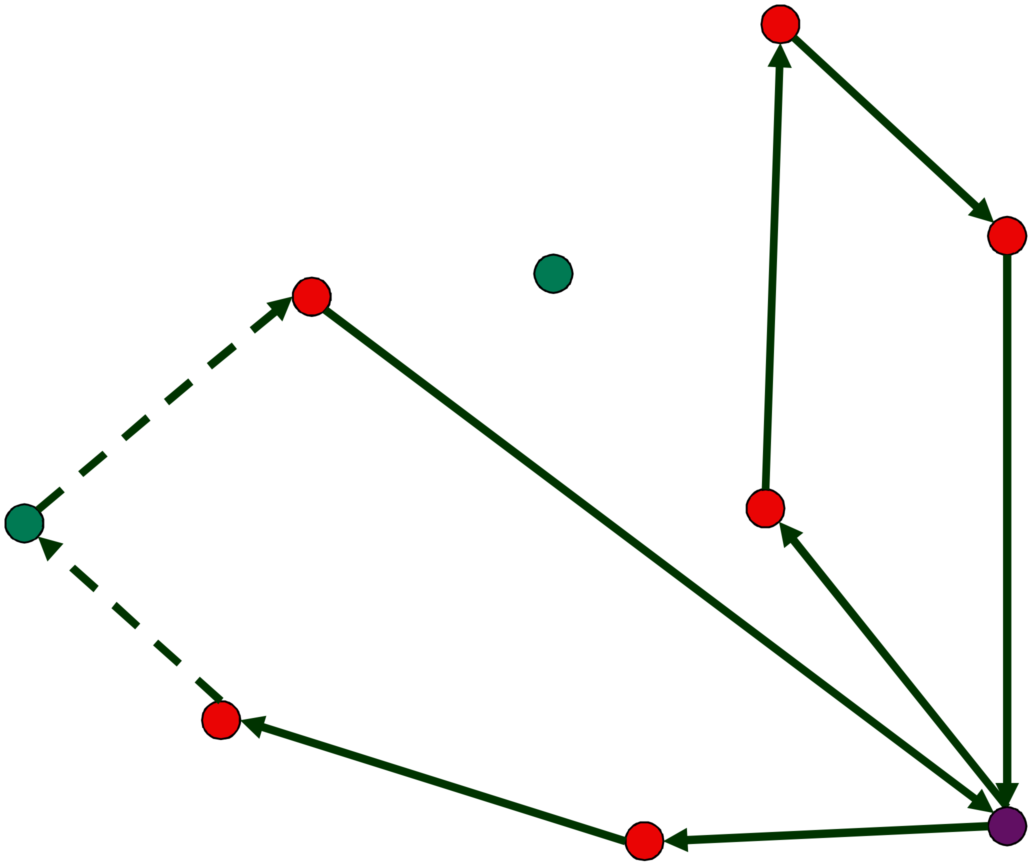
Regret



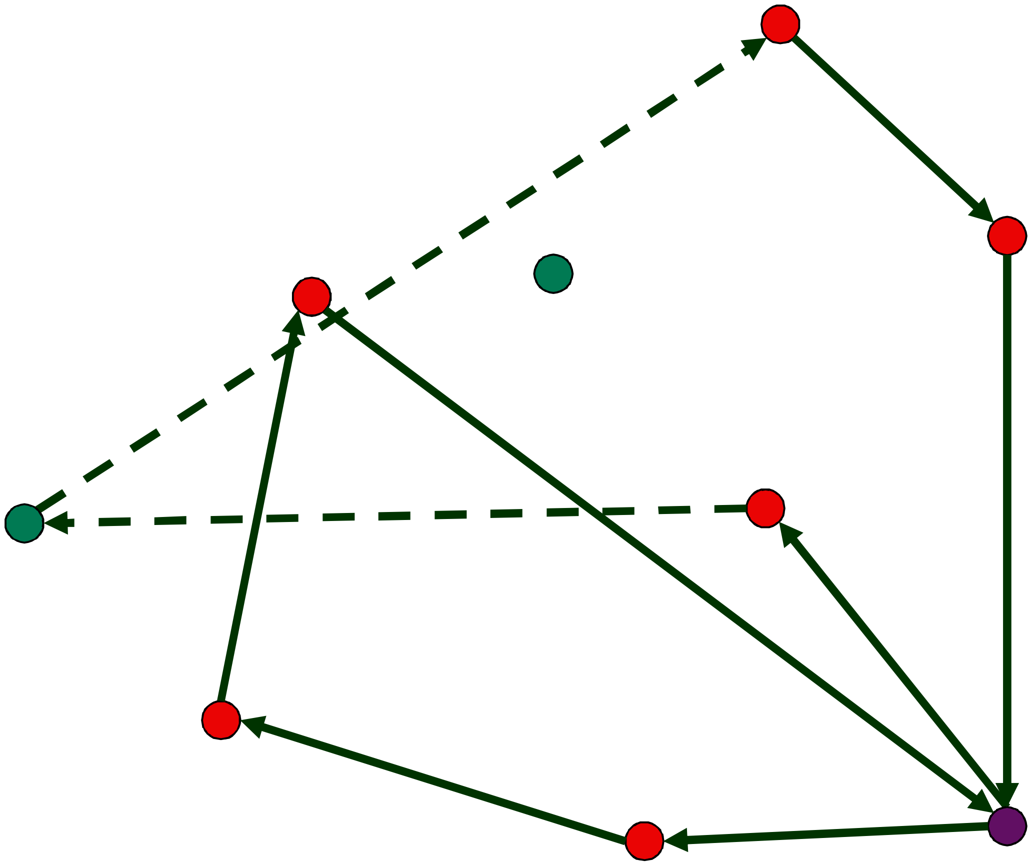
Regret



Regret



Regret



Regret

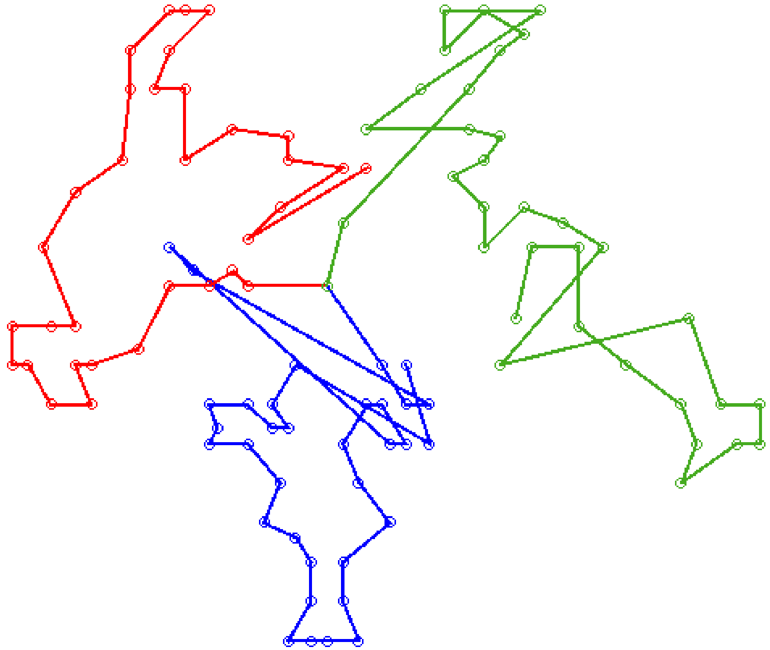
Regret = C(insert in 2nd-best route) – C(insert in best route)

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

K-Regret = $\sum_{k=1,K} (f(k,i) - f(1,i))$

Insert customer with maximum regret

Insertion with Regret



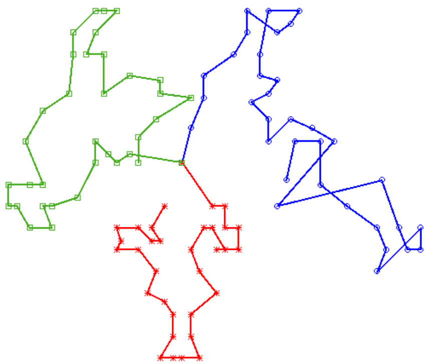
Seeds

Initialise each route with one (or more) customer(s)

- Indicates the general area where a vehicle will be
- May indicate time it will be there
  - Depends on time window width

Distance-based seeding

- Find the customer (s_1) most distant from the depot
- Find the customer (s_2) most distant from s_1
- Find the customer (s_3) mist distance from s_1, s_2
- ...
- Continue until all vehicles have a seed



Implementation

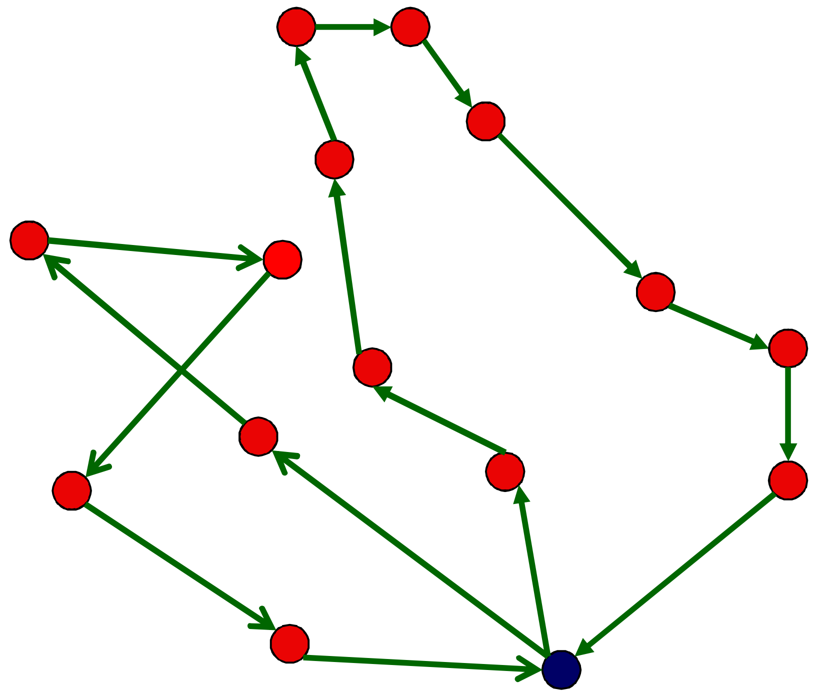
- Heart of algorithm is deciding which customer to insert next, and where
- Data structure of "Insert Positions"
  - legal positions to insert a customer
  - Must calculate cost of insert
  - Must ensure feasibility of insert
- After each modification (customer insert)
  - Add new insert positions
  - Update cost of affected insert positions
  - Check legality of all insert positions
  - *O(1) check important for efficiency*

Local Search

Improvement Methods

Local Search

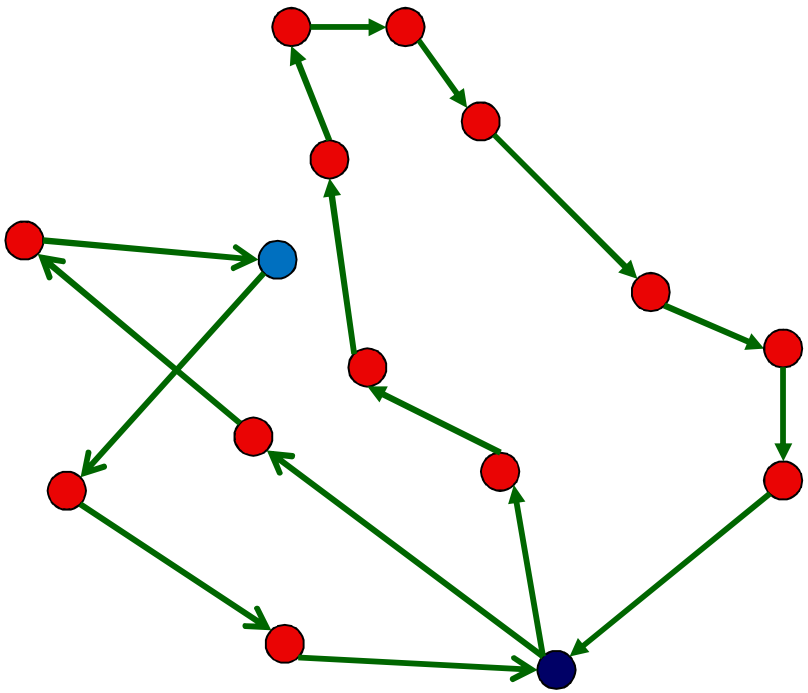
- Often defined using an “operator”



Improvement Methods

Local Search

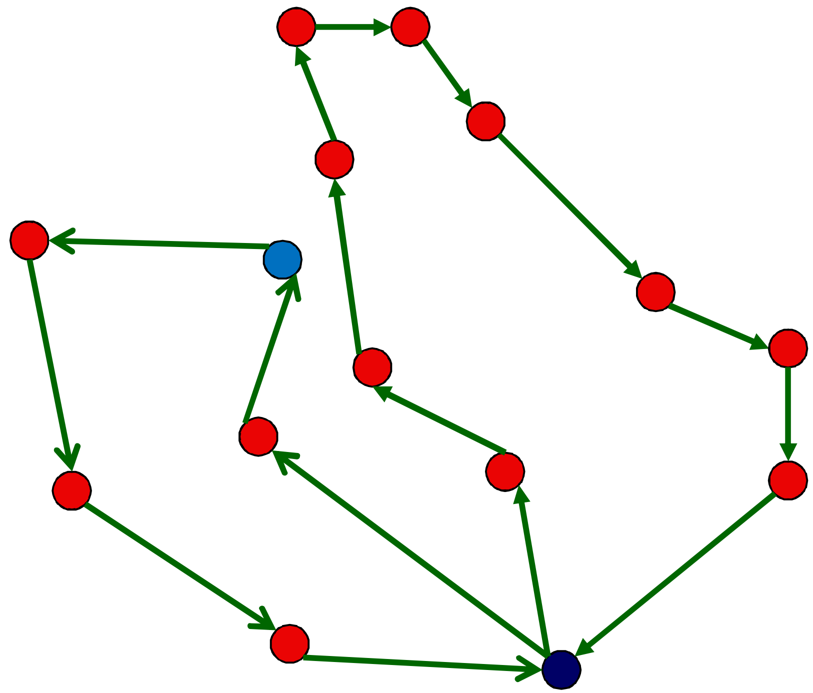
- Often defined using an “operator”
  - e.g. 1-move



Improvement Methods

Local Search

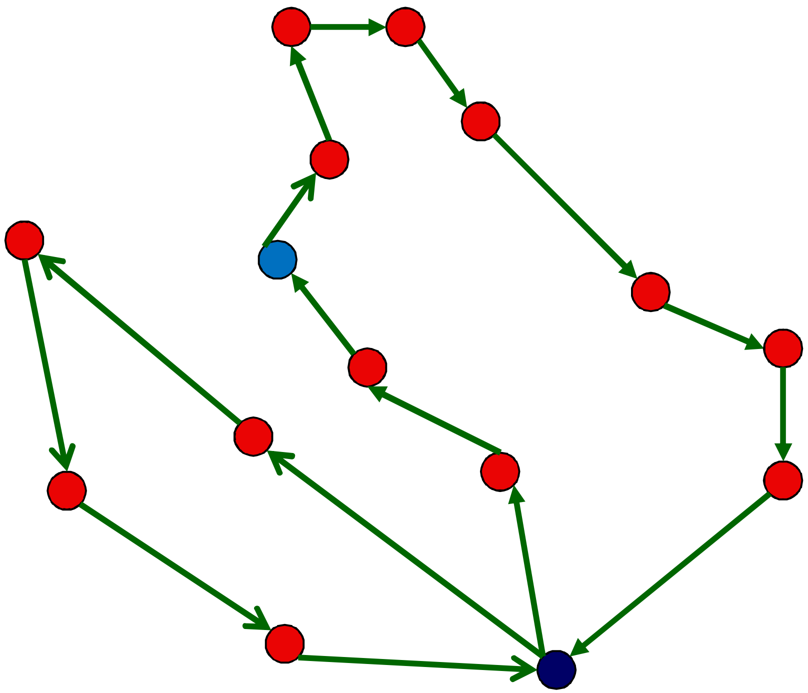
- Often defined using an "operator"
    - e.g. 1-move



Improvement Methods

Local Search

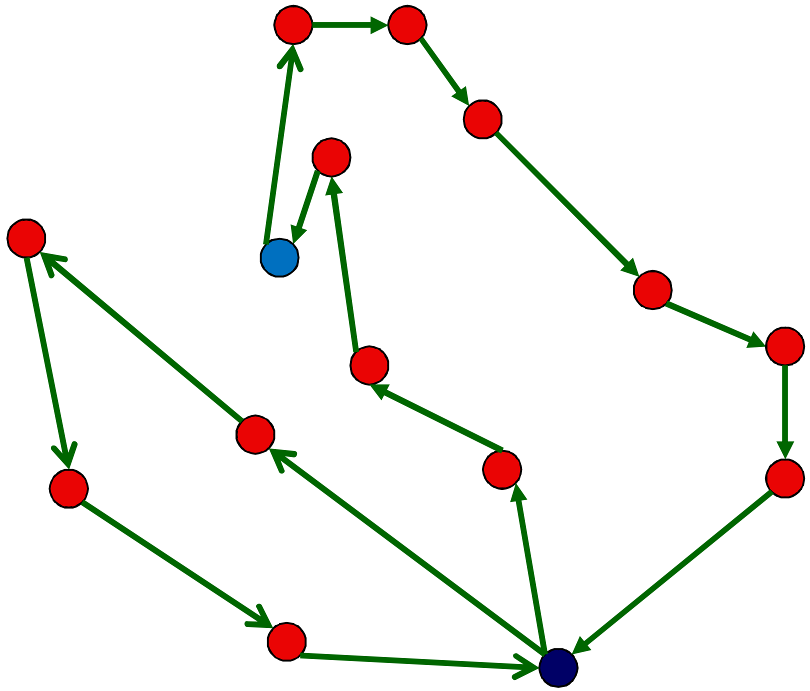
- Often defined using an “operator”
  - e.g. 1-move



Improvement Methods

Local Search

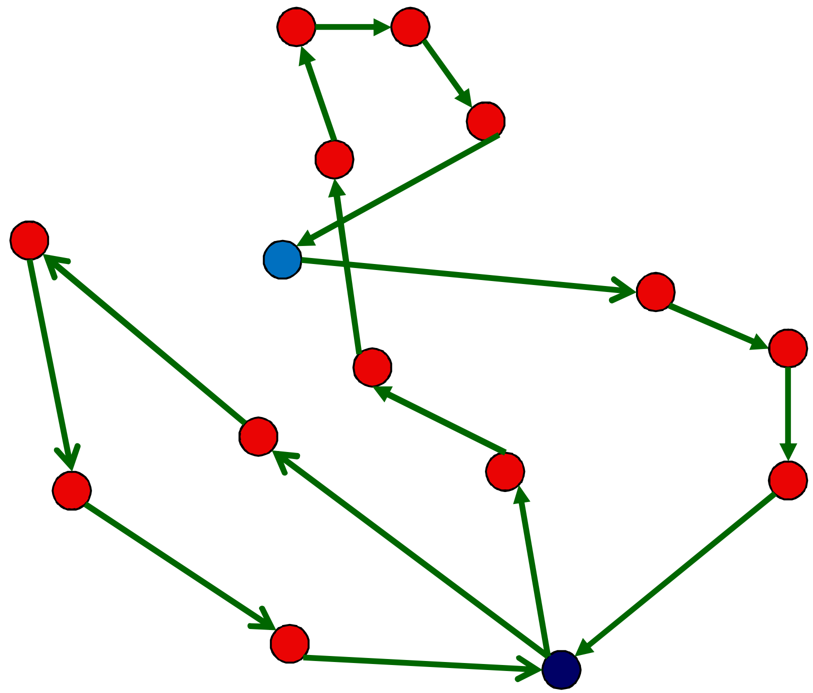
- Often defined using an “operator”
    - e.g. 1-move



Improvement Methods

Local Search

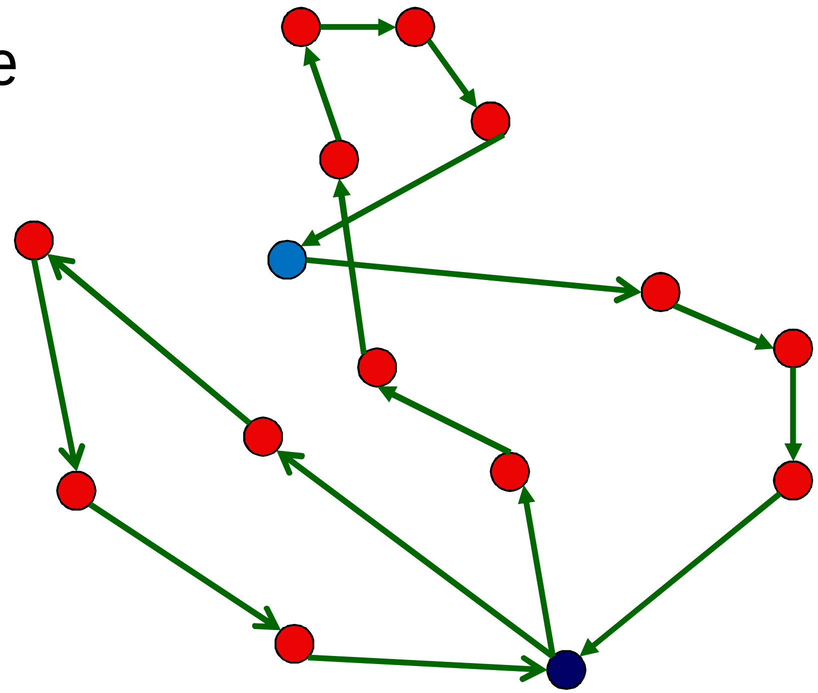
- Often defined using an “operator”
  - e.g. 1-move



Improvement Methods

Local Search

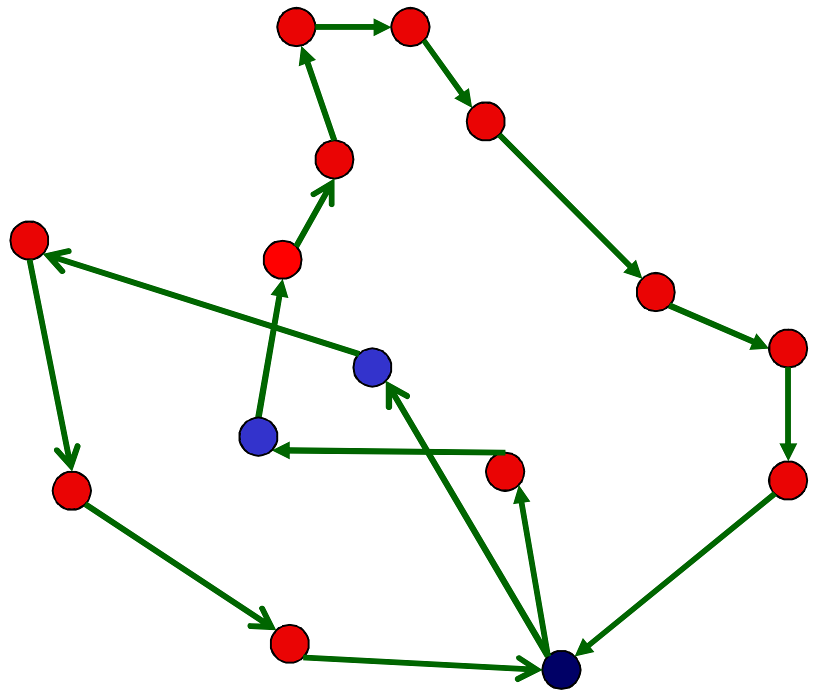
- Often defined using an "operator"
  - e.g. 1-move
- Solutions that can be reached using the operator termed the *neighbourhood*
- Local Search explores the neighbourhood of the current solution



Local Search

Other Neighbourhoods for VRP:

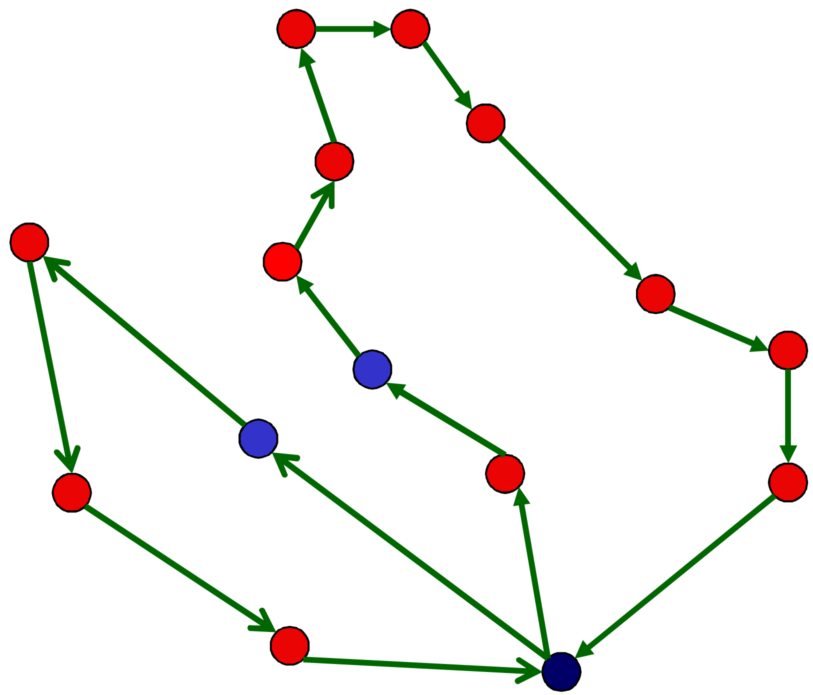
- Swap 1-1



Local Search

Other Neighbourhoods for VRP:

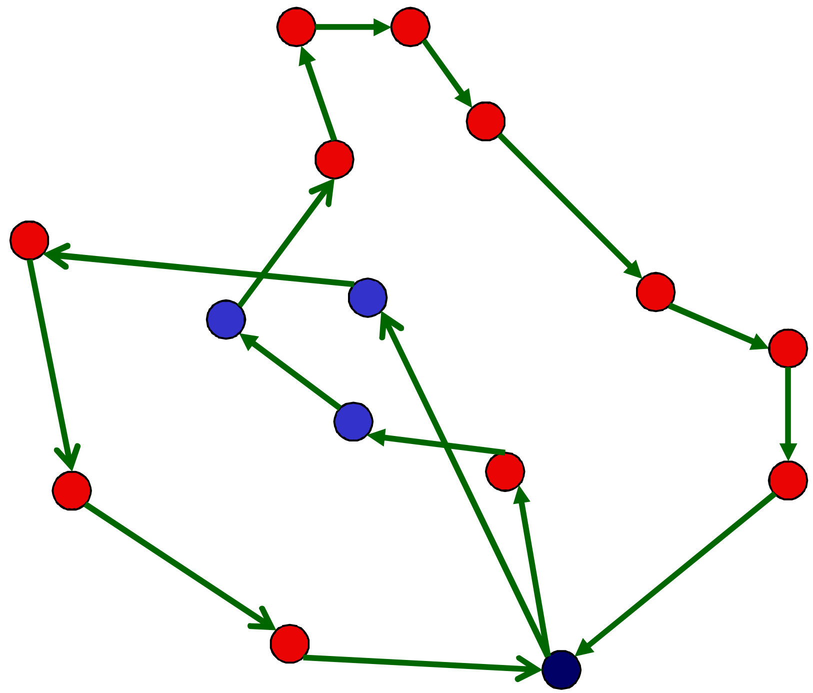
- Swap 1-1



Local Search

Other Neighbourhoods for VRP:

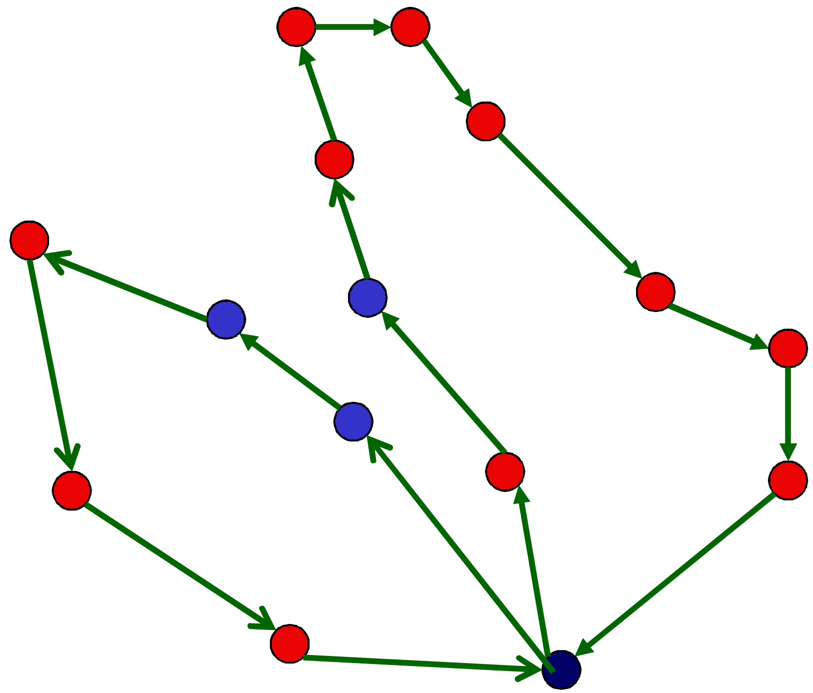
- Swap 2-1



Local Search

Other Neighbourhoods for VRP:

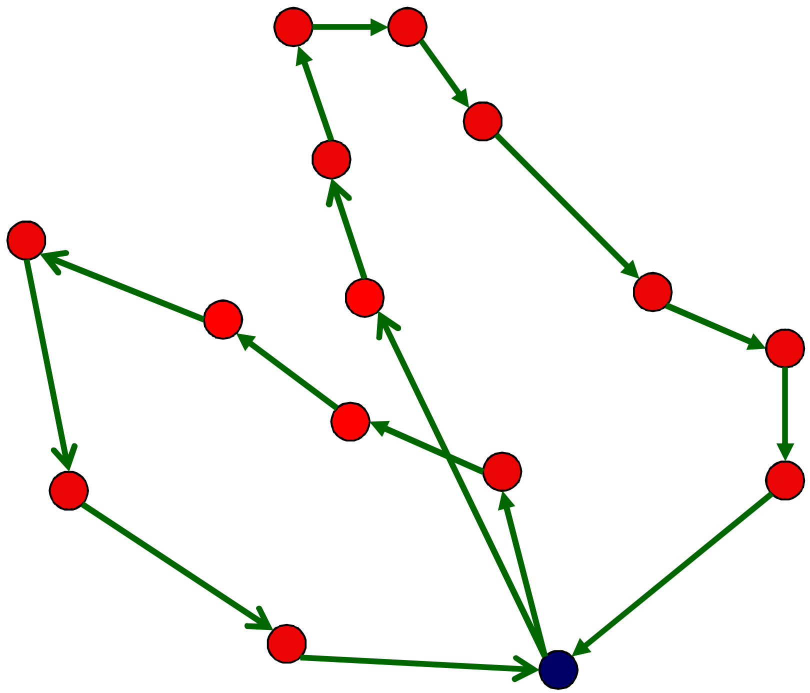
- Swap 2-1



Local Search

Other Neighbourhoods for VRP:

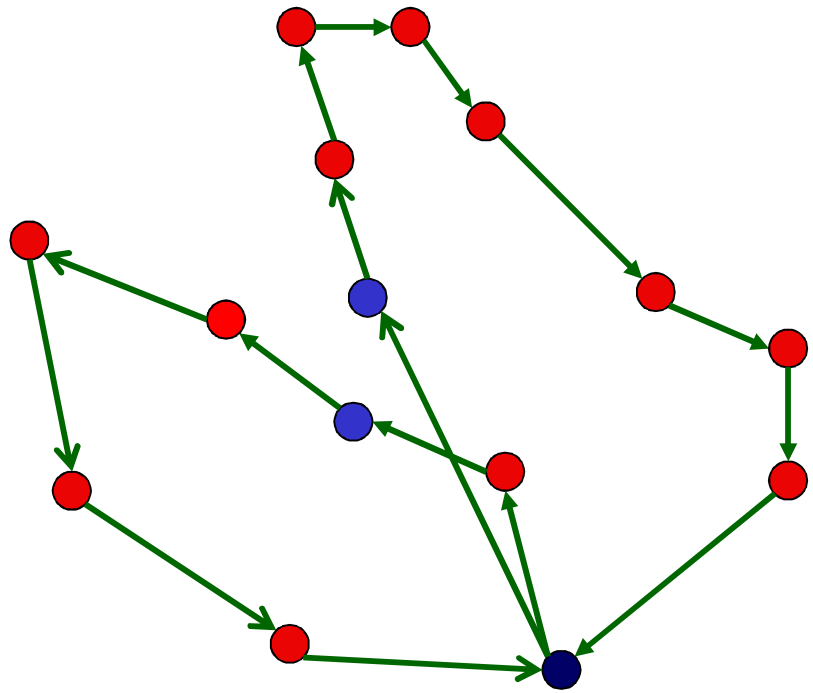
- Swap tails



Local Search

Other Neighbourhoods for VRP:

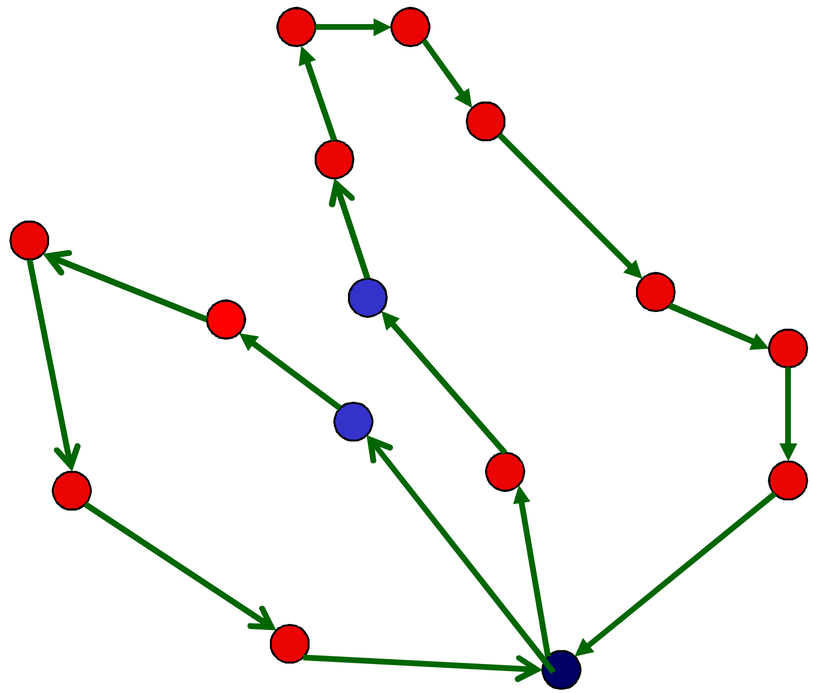
- Swap tails



Local Search

Other Neighbourhoods for VRP:

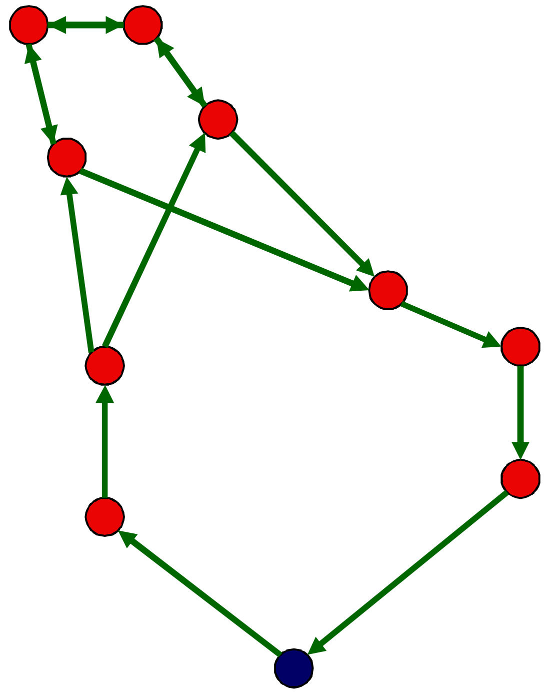
- Swap tails



Improvement Methods

2-opt (3-opt, 4-opt...)

- Remove 2 arcs
- Replace with 2 others



Improvement Methods

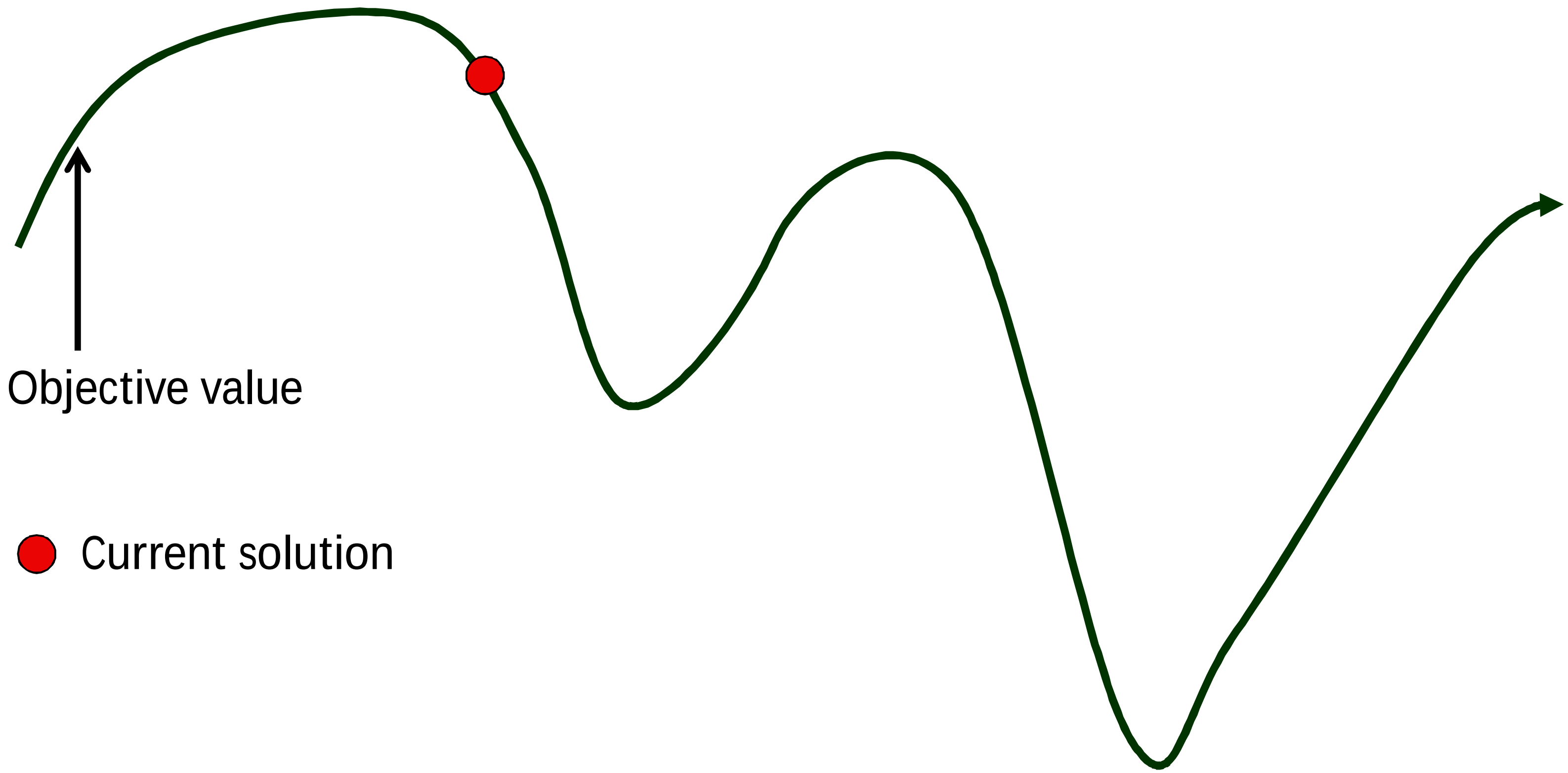
Or-opt

- Consider chains of length k
- k takes value 1 .. n / 2
- Remove the chain from its current position
- Consider placing in each other possible position
  - in forward orientation
  - and reverse orientation

- Very effective

Local Search

- Local minima



Local Search

Escaping local minima

Meta-heuristics

- Heuristic way of combining heuristics
- Designed to escape local minima

Local Search

Escaping local minima

- Define more (larger) neighbourhoods
  - 1-move (move 1 visit to another position)
  - 1-1 swap (swap visits in 2 routes)
  - 2-2 swap (swap 2 visits between 2 routes)
  - Tail exchange (swap final portion of routes)
  - 2-opt
  - Or-opt (all sizes 2 .. *n/2)*
  - 3-opt

Local Search

Variable Neighbourhood Search

- Consider multiple neighbourhoods
  - 1-move (move 1 visit to another position)
  - 1-1 swap (swap visits in 2 routes)
  - 2-2 swap (swap 2 visits between 2 routes)
  - 2-opt
  - Or-opt
  - Tail exchange (swap final portion of routes
  - 3-opt
- Explore one neighbourhood completely
- If no improvement found, advance to next neighbourhood
- When an improvement is found, return to level 1

Local Search

Variable Neighbourhood Search

- For new constraints/new problems, add new neighbourhoods

- E.g. Orienteering problem
  - o New neighbourhoods:
    - Unassign 1 customer (i.e. do not visit)
    - Unassign clusters of customer (e.g. sequences of customers)
    - Insert clusters of unassigned customers

Local Search

Many Meta-heuristics have been tried

- Simulated Annealing
- Tabu Search
- Genetic Algorithms
- Ants
- Bees
- Particle Swarms

- Large Neighbourhood Search

Large Neighbourhood Search

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

- Destroy part of the solution
  - Remove visits from the solution
- Re-create solution
  - Use favourite construct method to re-insert customers
- 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, 2006

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

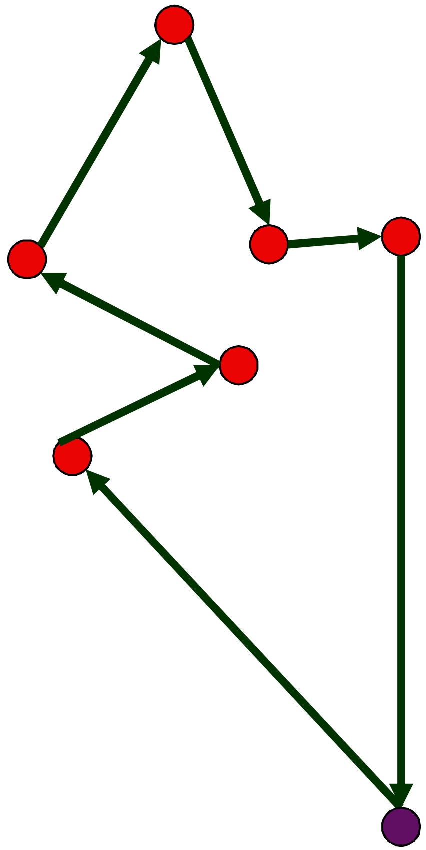
- Remove some visits
- Move them to the “unassigned” lists

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Remove a sequence of visits

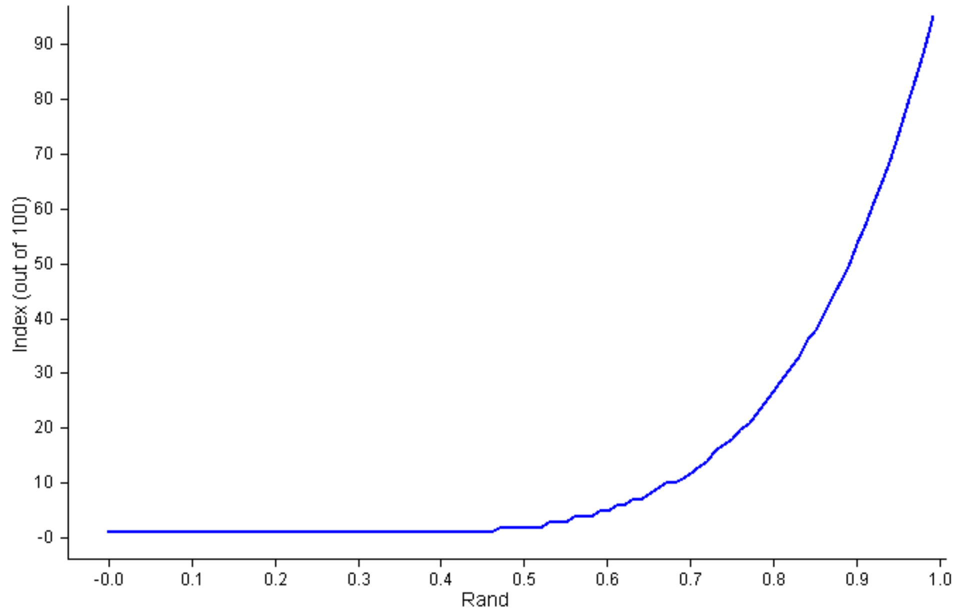


Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Choose longest (worst) arc in solution
  - Remove visits at each end
  - Remove nearby visits
- Actually, choose r^{th} worst
- $r = n * (\text{uniform}(0,1))^{y}$
- $y \sim 6$
  - $0.5^6 = 0.016$
  - $0.9^6 = 0.531$



Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Dump 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

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

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
- Actual number is uniform rand (5, *max*)

Large Neighbourhood Search

Re-create solution

- Systematic search
  - Smaller problem, easier to solve
  - Can be very effective
  - E.g.: CP Backtracking search
  - Constraint: objective must be less than current
  - (Implicitly) Look at all reconstructions
  - Backtrack as soon as a better sol is found
  - Backtrack anyway after *too many* failures

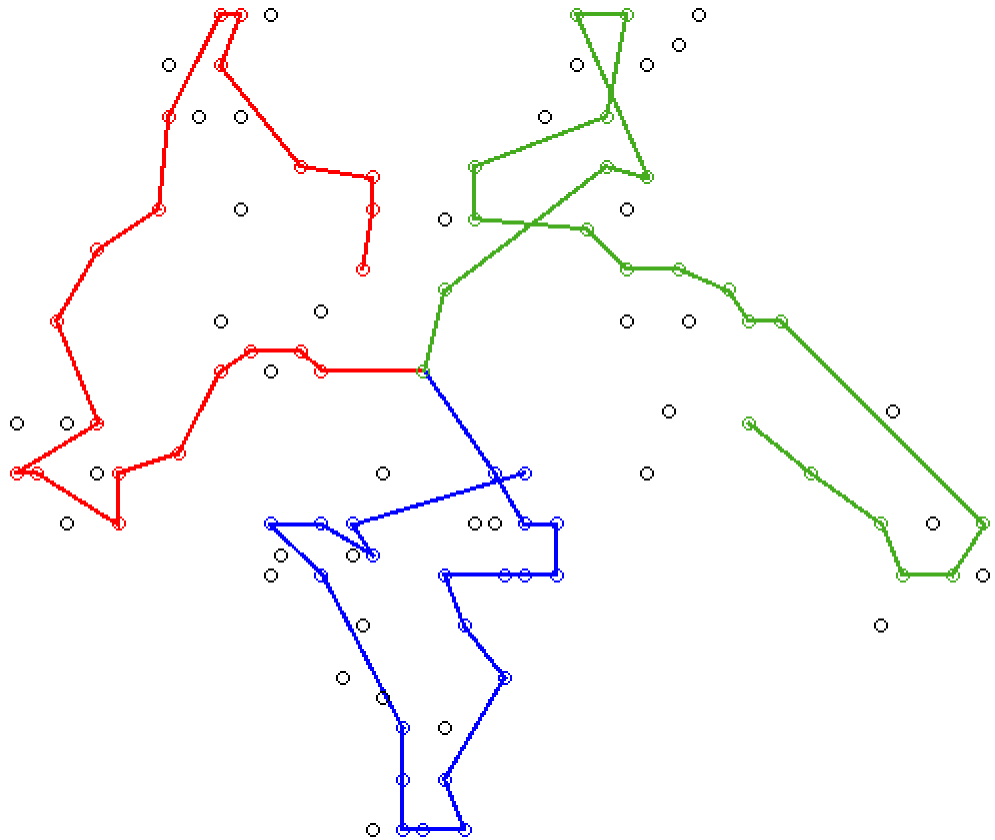
Large Neighbourhood Search

Re-create solution

- Use your favourite insert method

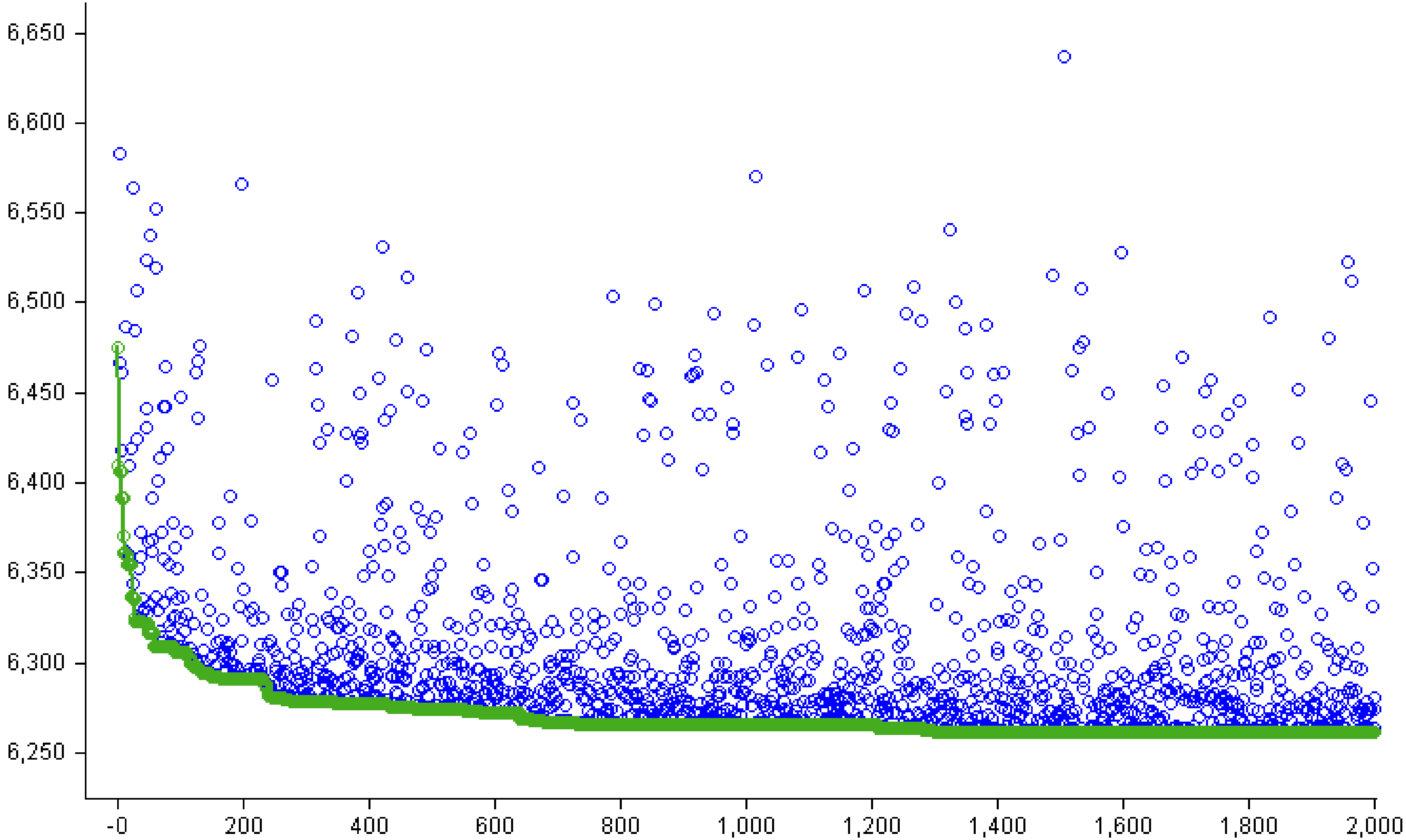
- Better still, use a portfolio
  - Ropke: Select amongst
    - Minimum Insert Cost
    - Regret
    - 3-regret
    - 4-regret

Large Neighbourhood Search



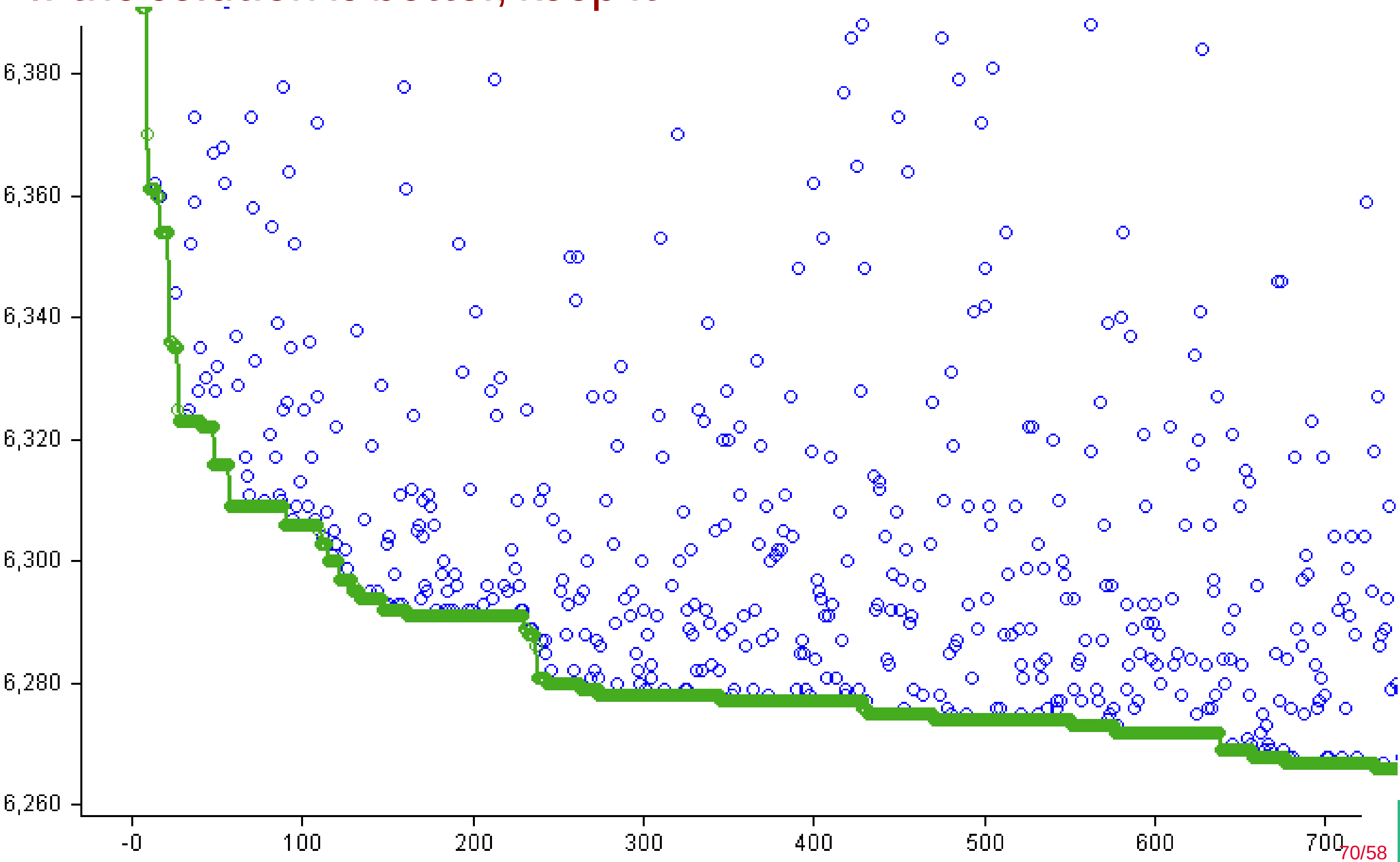
Large Neighbourhood Search

- If the solution is better, keep it



Large Neighbourhood Search

- If the solution is better, keep it

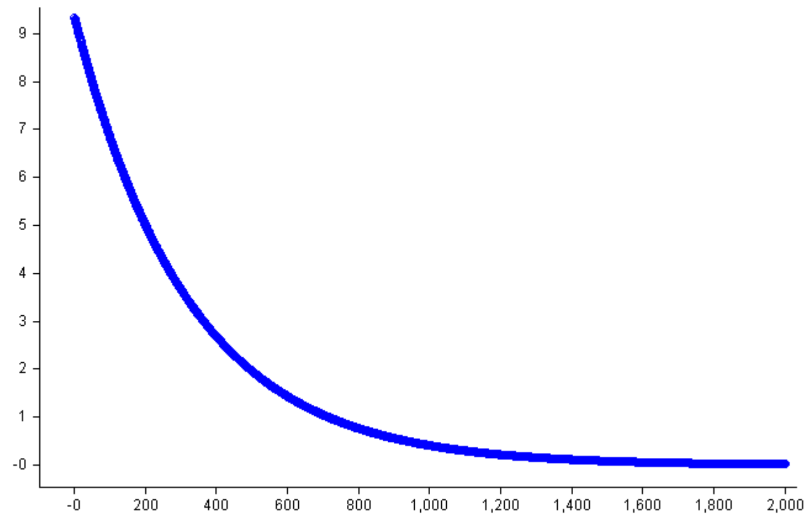


Large Neighbourhood Search

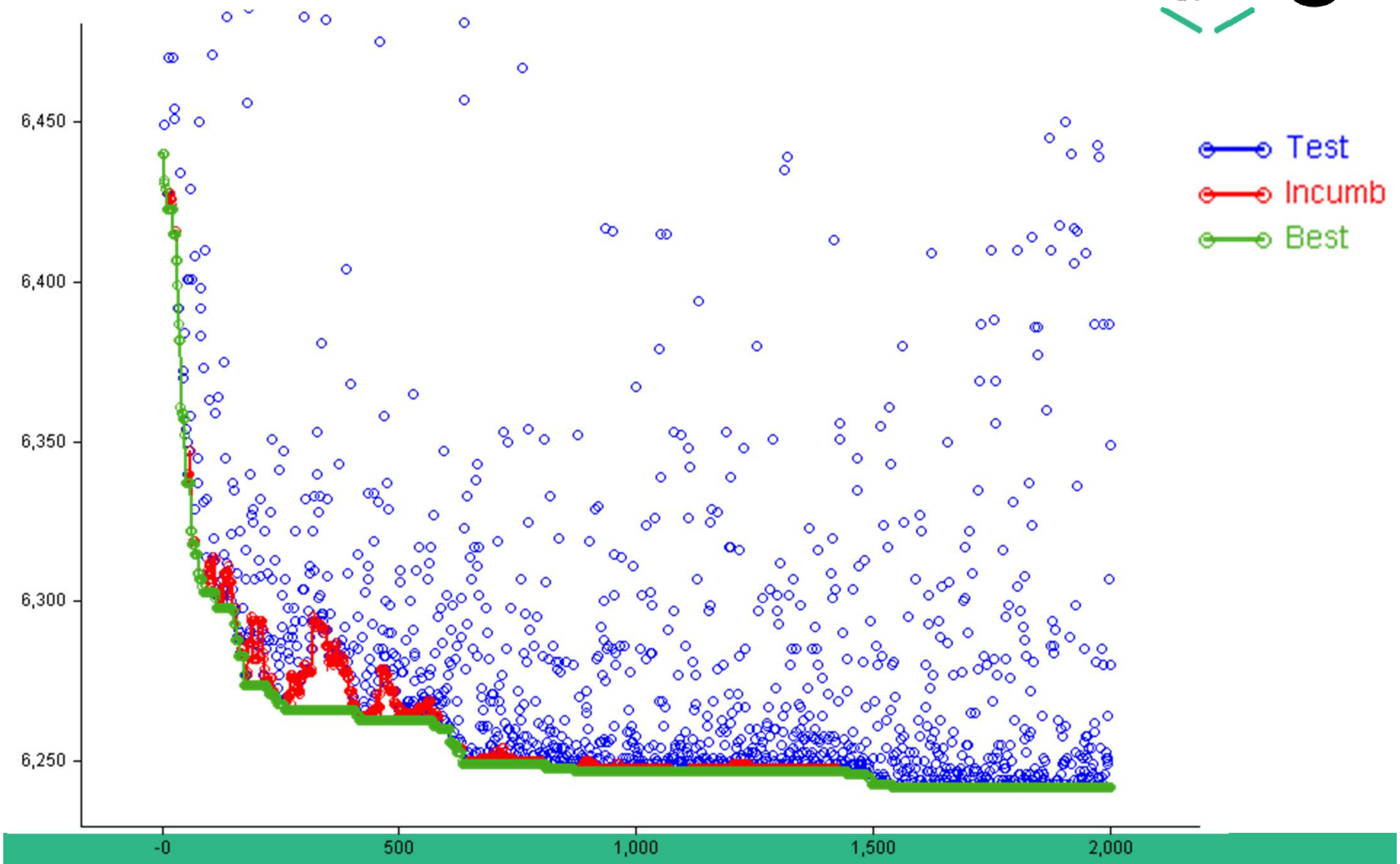
- If the solution is better, keep it
- Can use Hill-climbing
- Can use Simulated Annealing
- Can use Threshold Annealing
- ...

Large Neighbourhood Search

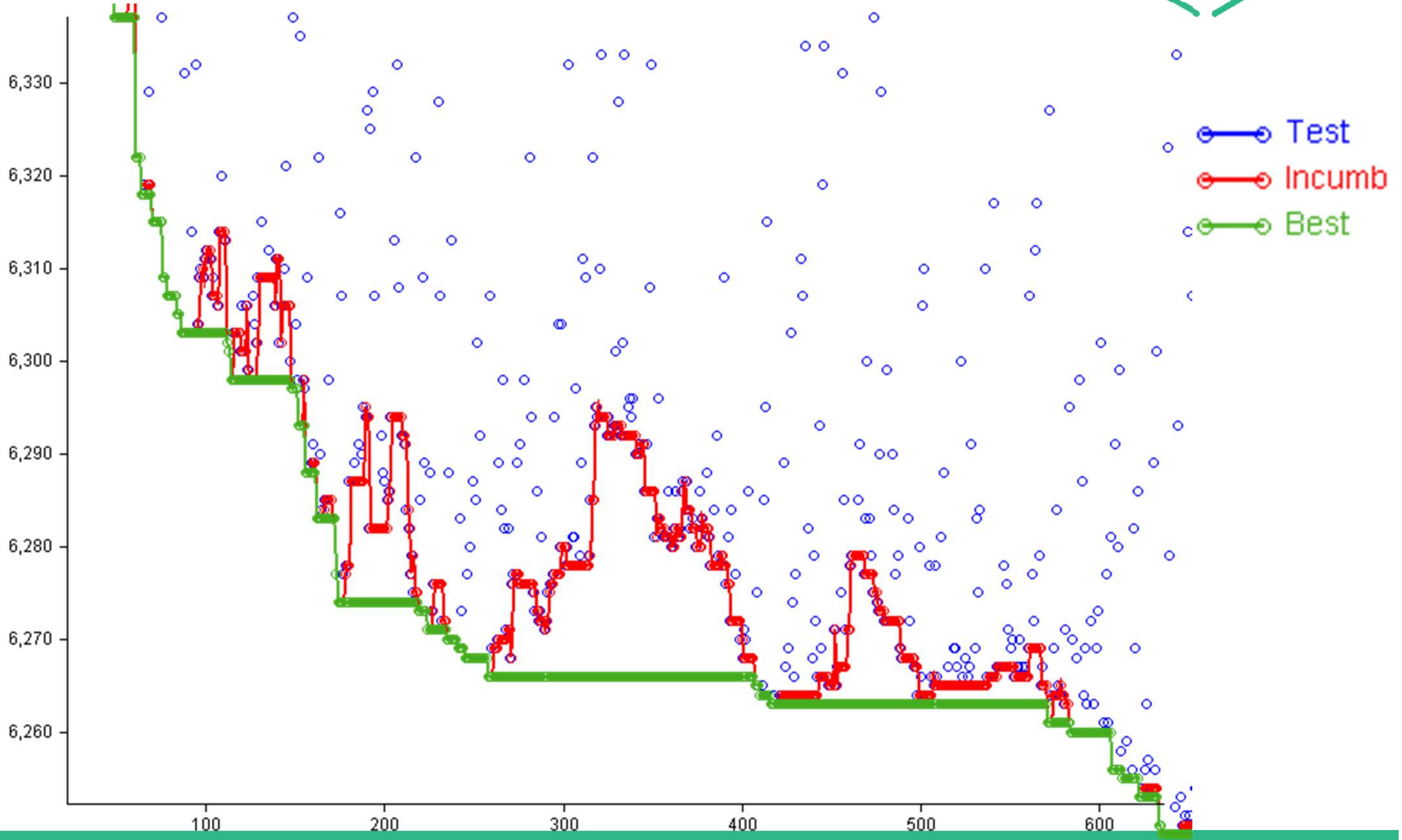
$$P(accept\,increase\,\Delta) = e^{-\Delta/T}$$



Large Neighbourhood Search



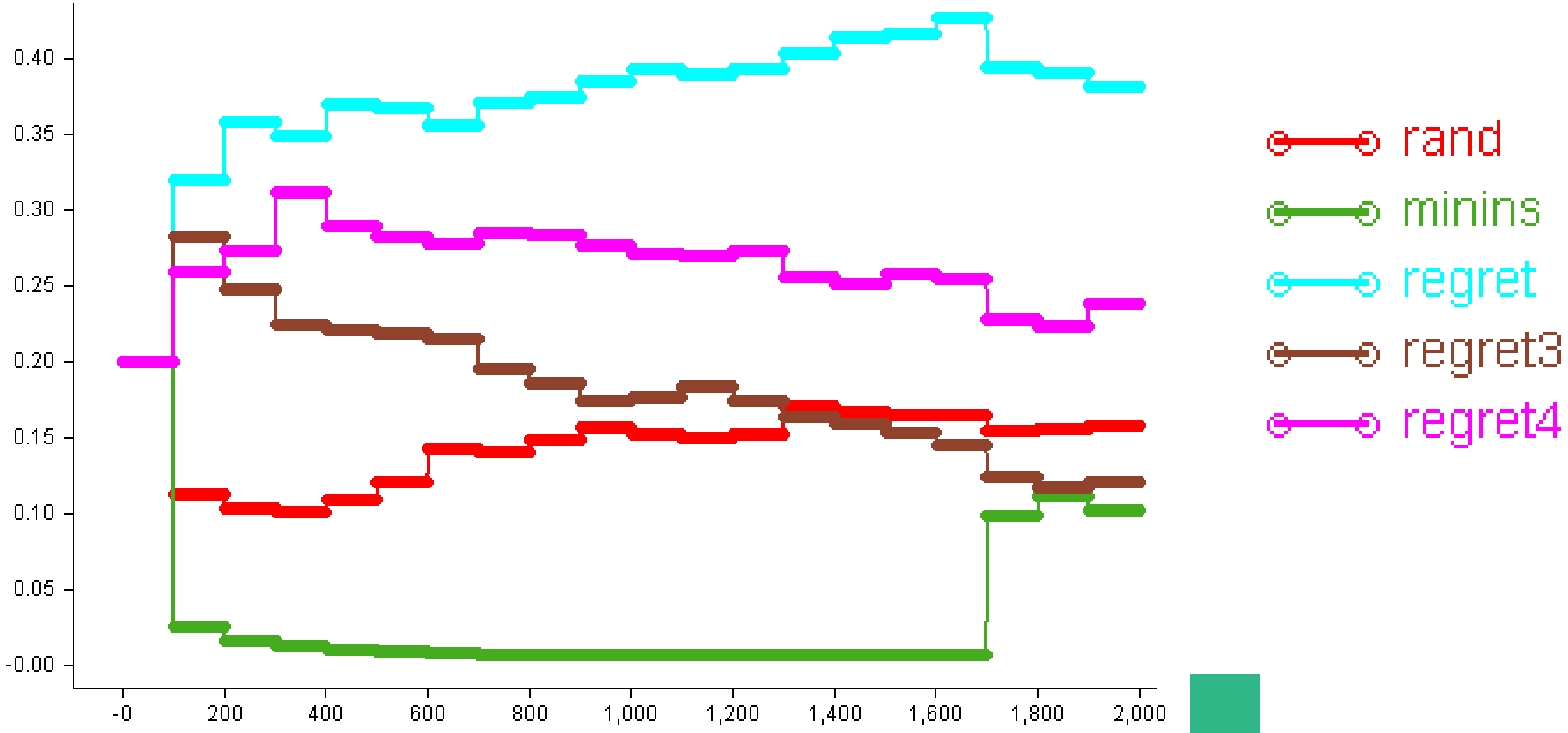
Large Neighbourhood Search



Large Neighbourhood Search

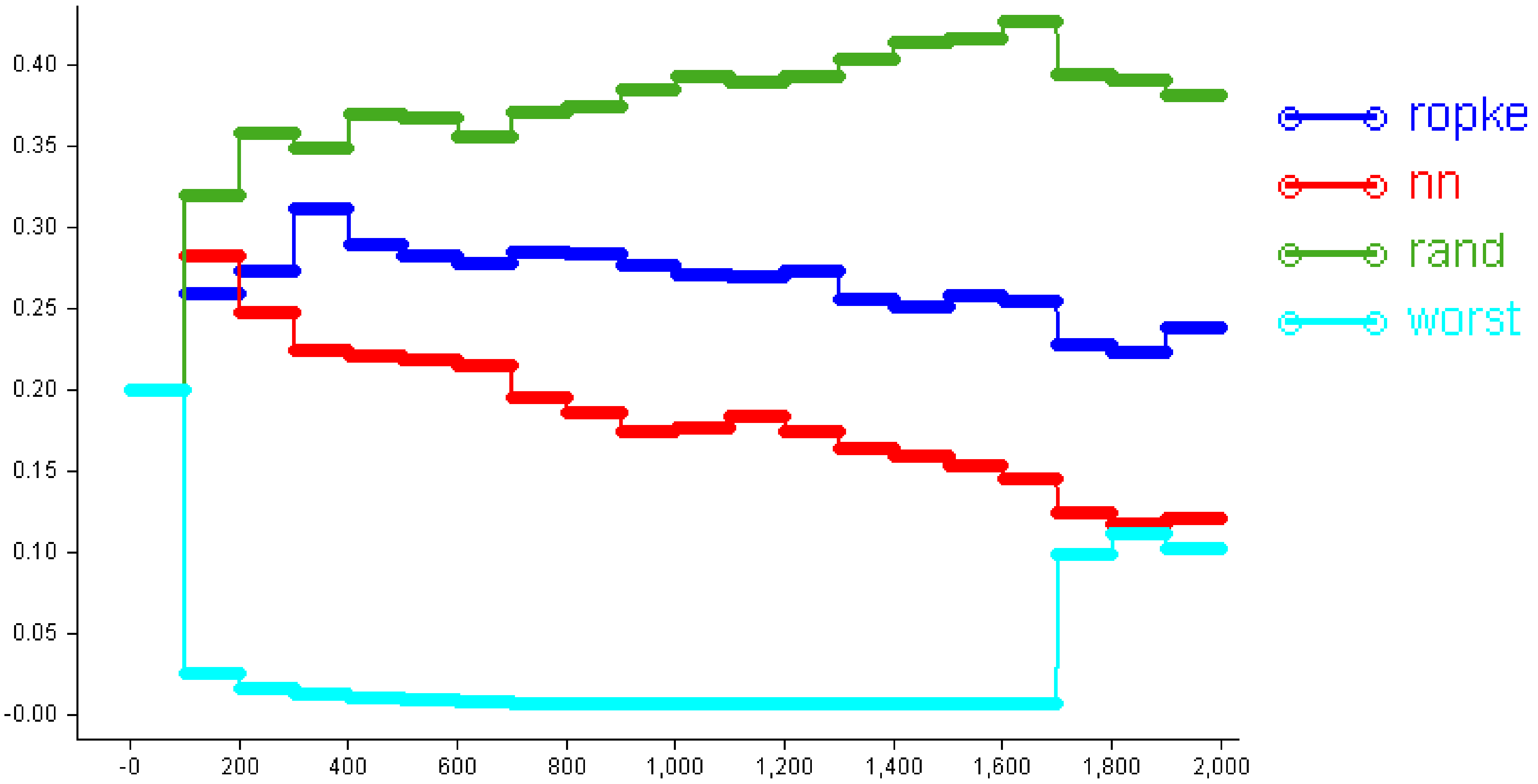
Adaptive

– Ropke adapts choice based on prior performance
  – “Good” methods are chosen more often



Large Neighbourhood Search

Adapting Select 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

Solution Methods

Summary so far:

- Introduced several successive insertion construction methods
  - Various ways to choose the next visit to insert
  - Various ways to choose where to insert
- Described two successful metaheuristics
  - Variable Neighbourhood Search
  - Large Neighbourhood Search

Solution Methods

What's wrong with that?

- New constraint → new code
  - Often right in the core
- New constraints interact
  - e.g. Multiple time windows mess up duration calculation
- Code is hard to understand, hard to maintain

Solution Methods

An alternative:

Constraint Programming

Constraint Programming

CP offers a language for representing problems

- Decision variables
- Constraints on variables

Also offers techniques for solving the problems

- Systematic search
- Heuristic Search

CP101

Variables are represented by their *domain*

- (Usually finite) set of feasible values
- E.g $x \in [0,100]$ or $x \in [0,1,3..15,16,18,55..99]$

Constraints link variables

- $x \leq 4y + 6z$
- $x^2 + y^2 = z^2$
- Cardinality (X, 1, 4, 5)
  (In the set *X* the value '1' occurs at least 4 times, and no more than 5 times)
- AllDifferent (X) (All values in X are different)
- DriverBreak (30,120,240)
  (A break of 30 minutes must be inserted after 120 minutes but no later than 240 minutes after start of route)

CP101

Propagators (efficiently) enforce constraints

- Wake when the domain of a linked variable is changed
- For each value in each variable
  - Ensure there is a set of feasible values of other variables that *supports* that value – e.g.

$$x < y$$

x = [3,5,7,9]

y = [2,4,6,8]

  - The value ‘9’ in *x* has no support in *y*
  - The value ‘2’ in *y* has no support in *x*
  - After propagation: x = [3,5,7]

    y = [4,6,8]

CP101

Eg Mutual Exclusion constraint in VRP

- (If any visit from the set D is assigned, then no others can be)

- Uses ‘isAssigned’ var
  - Domain [0,1]
- Attach propagator to the ‘isAssigned’ variable for each of the visits
- Propagator wakes when ‘isAssigned’ is bound to 1 for any visit
- Propagates by binding isAssigned to ‘0’ for remaining visits.

CP101

- Typical execution:
  - Establish choice point (store all current domains)
  - Choose variable to instantiate
  - Choose value to assign, and assign it
  - Propagations fire until a *fixed point* is achieved, or an inconsistency is proved (empty domain)
  - If inconsistent,
    - Backtrack (restore to choicepoint)
    - Remove offending value from the variable's domain
  - Repeat until all variables are bound (assigned)
  - For complete search, store sol, then act like inconsistent

CP101

‘Choose a variable to assign, choose value to assign’

- Very good fit for constructive route creation
- After each insert, propagators fire
- New variable domains give look-ahead to feasible future insertions
- Constraints guide insertion process

Step-to-new-solution does not work as well

- Local move operators can only use CP as a rule checker
    - Do not leverage full power of CP

Expressive Language (e.g. Minizinc)

```
string: Name;

% Customers
int: NumCusts;
set of int: Customers =
                    1..NumCusts;

% Locations
int: NumLocs = NumCusts + 1;
set of int: Locations =
                     1..NumLocs;

% Vehicles
int: NumVehicles;
int: NumRoutes = NumVehicles;
set of int: Vehicles =
                 1..NumVehicles;

% Location data
array[Locations] of float: locX;
array[Locations] of float: locY;
array [Locations, Locations] of
int: dist;
```

```
% Decision variables
var int: obj;
array[Visits] of var Visits:
   routeOf;
array[Visits] of var Visits:
   succ;
array[Visits] of var [0,1]:
   isAssigned;

constraint alldifferent (succ);
constraint circuit (succ);

constraint
  obj = sum (i in Visits)
    (dist[Loc[i],Loc[succ[i]]]);

constraint
sum (i in Visits,j = routeOf[i])
   (demand[i]) < j)
       for j in Vehicles;
```

Constraint Programming for the VRP

Constraint Programming

Advantages:

- Expressive language for formulating constraints
- Each constraint encapsulated
- Constraints interact naturally
- Constraints guide construction

Disadvantages:

- Can be slow
- No fine control of solving
  - (unless you use a low-level library like *gecode*)

Constraint Programming

Two ways to use constraint programming

- Rule Checker
- Properly

Rule Checker:

- Use favourite method to create/improve a solution
- Check it with CP
  - Very inefficient.

A CP Model for the VRP

Vocabulary

- A *solution* is made up of *routes* (one for each vehicle)
- A *route* is made up of a sequence of *visits*
- Some visits serve a customer (*customer visit*)

(Some tricks)

- Each route has a “start visit” and an “end visit”
- Start visit is first visit on a route – location is depot
- End visit is last visit on a route – location is depot
- Also have an additional route – the unassigned route
  - Where visits live that cannot be assigned

Model

A (rich) vehicle routing problem

- n customers (fixed in this model)
- v vehicles (fixed in this model)
- $m = v+1$ one route per vehicle plus "unassigned" route
- fixed locations
  - where things happen
  - one for each customer + one for (each?) depot
- c commodities (e.g. weight, volume, pallets)
  - Know demand from each customer for each commodity
- Know time between each location pair
- Know cost between each location pair
  - Both obey triangle inequality

Referencing

Sets

- $N = \{1 \;..\; n\}$ – customers
- $V = \{1 \;..\; v\}$ – vehicles/real routes
- $R = \{1 \;..\; m\}$ - routes include 'unassigned' route
- $S = \{n+1 \;..\; n+m\}$ – start visits
- $E = \{n+m+1 \;..\; n+2m\}$ – end visits
- $V = N \cup S \cup E$ – all visits
- $V^S = N \cup S$ – visits that have a sensible successor
- $V^E = N \cup E$ – visits that have a sensible predecessor

Referencing

Customers

- Each customer has an index in $N = \{1..n\}$
- Customers are ‘named’ in CP by their index

Routes

- Each route has an index in $R = \{1..m\}$
- Unassigned route has index m
- Routes are ‘named’ in CP by their index

Visits

- Customer visit index same as customer index
- Start visit for route k has index `n + k ;` aka $\mathtt{start}_k$
- End visit for route k has index `n + m + k ;` aka $\mathtt{end}_k$

Data

We know (note uppercase)

- V_i The 'value' of customer i
- D_{ik} Demand by customer i for commodity k
- E_i Earliest time to start service at i
- L_i Latest time to start service at i
- Q_{jk} Capacity of vehicle j for commodity k
- T_{ij} Travel time from visit i to visit j
- C_{ij} Cost (w.r.t. objective) of travel from i to j

Basic Variables

Successor variables: s_i

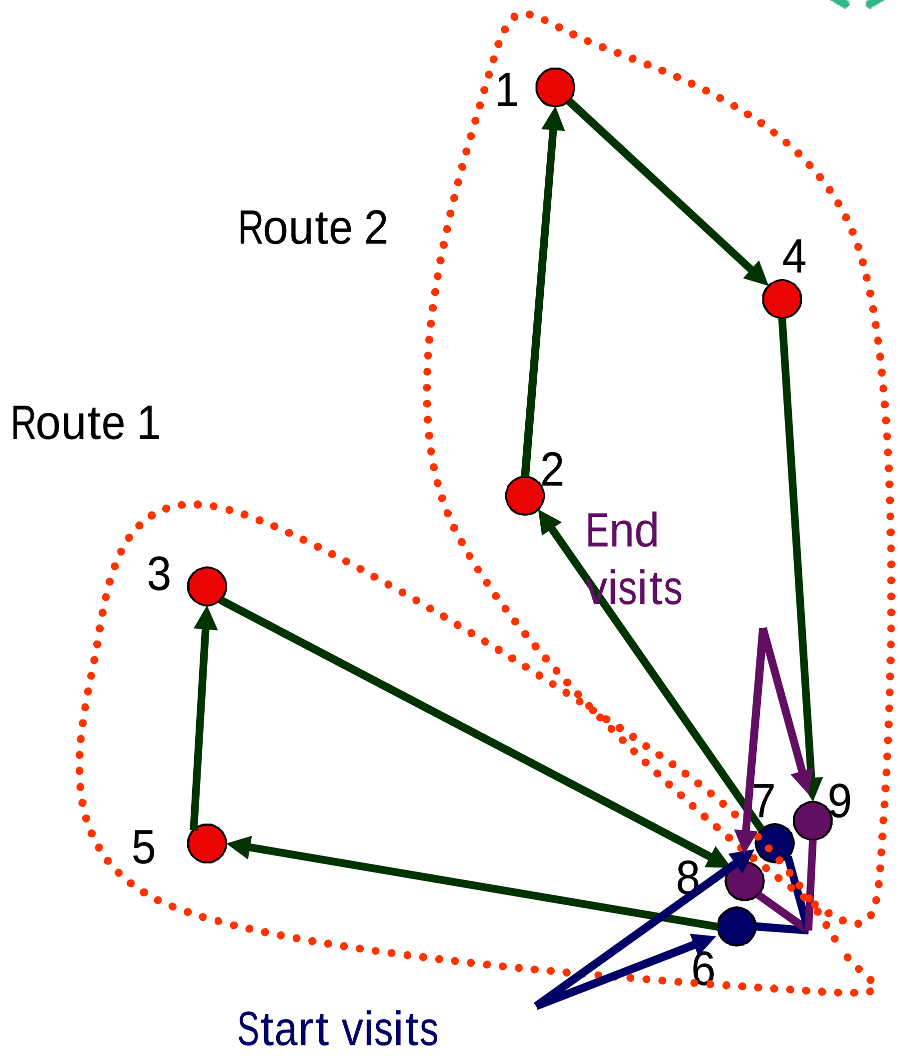
- s_i gives direct successor of i, i.e. the index of the next visit on the route that visits i
- $s_i \in V^E$ for i in V^S $s_i = 0$ for i in E

Predecessor variables p_i

- p_i gives the index of the previous visit in the route
- $p_i \in V^S$ for i in V^E $p_i = 0$ for i in S
- Redundant – but empirical evidence for its use
- Route variables r_i
- r_i gives the index of the route (vehicle) that visits i
- $r_i \in R$

Example

i	s_i	p_i	r_i
1	4	2	2
2	1	7	2
3	8	5	1
4	9	1	2
5	3	6	1
6	5	0	1
7	2	0	2
8	0	3	1
9	0	4	2



Other variables

Accumulation Variables

- q_{ik} Quantity of commodity k after visit i
- c_i Objective cost getting to i

- For problems with time constraints
- a_i Arrival time at i
- t_i Start time at i (time service starts)
- d_i Departure time at i

- Actually, only t_i is required, but others allow for expressive constraints

What can we model?

- Basic VRP
- VRP with time windows
- Multi-depot
- Heterogeneous fleet
- Open VRP (vehicle not required to return to base)
  - Requires *anywhere* location
  - Route end visits located at *anywhere*
  - distance `i` → *anywhere* = 0
- Compatibility
  - Customers on different / same vehicle
  - Customers on/not on given vehicle
- Pickup and Delivery problems

What can we model?

- Variable load/unload times
  - by changing departure time relative to start time
- Dispatch time constraints
  - e.g. limited docks
  - s_i for i in S is load-start time
- Depot close time
  - Time window on end visits
- Fleet size and mix
  - Add lots of vehicles
  - Need to introduce a ‘fixed cost’ for a vehicle
  - C_{ij} is increased by fixed cost for all $i \in S$, all $j \in N$

What can't we model

- Can't handle dynamic problems
  - Fixed domain for `s, p, r` vars
- Can't introduce new visits post-hoc
  - E.g. optional driver break must be allowed at start
- Can't handle multiple visits to same customer
  - 'Larger than truck-load' problems
  - If qty is fixed, can have multiple visits / cust
  - Heterogeneous fleet is a pain
- Can handle time- or vehicle-dependent travel times/costs with mods
- Can handle Soft Constraints with mods

Objective

Want to minimize

- sum of objective (c_{ij}) over used arcs, plus
- value of unassigned visits

minimize

$$\sum_{i \in E} c_i + \sum_{i \mid r_i = 0} v_i$$

Basic constraints

Path (S, E, { s_i | $i \in V$})
AllDifferent ({ p_i | $i \in V^E$})

Accumulate obj. $$c_{s_i} = c_i + C_{i,s_i} \quad \forall i \in V^S$$

Accumulate time $$a_{s_i} = d_i + T_{i,s_i} \quad \forall i \in V^S$$

Time windows $$t_i \geq a_i \quad \forall i \in V$$

$$t_i \leq L_i \quad \forall i \in V$$

$$t_i \geq E_i \quad \forall i \in V$$

$$t_i = 0 \quad \forall i \in S$$

Constraints

- Load

$$q_{s_i k} = q_{ik} + Q_{s_i k} \quad \forall i \in V^S, k \in C$$

$$q_{ik} \le Q_{r_i k} \quad \forall i \in V, k \in C$$

$$q_{ik} \ge 0 \quad \forall i \in V, k \in C$$

$$q_{ik} = 0 \quad \forall i \in S, k \in C$$

- Consistency

$$s_{p_i} = i \quad \forall i \in V^S$$

$$p_{s_i} = i \quad \forall i \in V^E$$

$$r_i = r_{s_i} \quad \forall i \in V^S$$

$$r_{n+k} = k \quad \forall k \in M$$

$$r_{n+m+k} = k \quad \forall k \in M$$

Subtour elimination

- Most CP libraries have built-ins
  - MiniZinc: ‘circuit’
  - Comet: ‘circuit’
  - ILOG: Path constraint

Propagation – Cycles

'Path' constraint

- Propagates subtour elimination
- Also propagates cost

- `path (S, E, succ, P, z)`
  - `succ` array implies path
  - ensures path from nodes in *S* to nodes in *T* through nodes in *P*
  - variable *z* bounds cost of path
  - cost propagated incrementally based on shortest / longest paths

Large Neighbourhood Search revisited

Large Neighbourhood Search

Destroy & Re-create

- Destroy part of the solution
  - Remove visits from the solution
- Re-create solution
  - Use insert method to re-insert customers
  - Different insert methods lead to new (better?) solutions
- If the solution is better, keep it
- Repeat

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

In CP terms, this means:

- Relax some variable assignments

In CP-VRP terms, this means

- Relax some *routeOf* and *successor* assignments

Large Neighbourhood Search

Re-create solution

- Use insert methods
- Uses full power of CP propagations

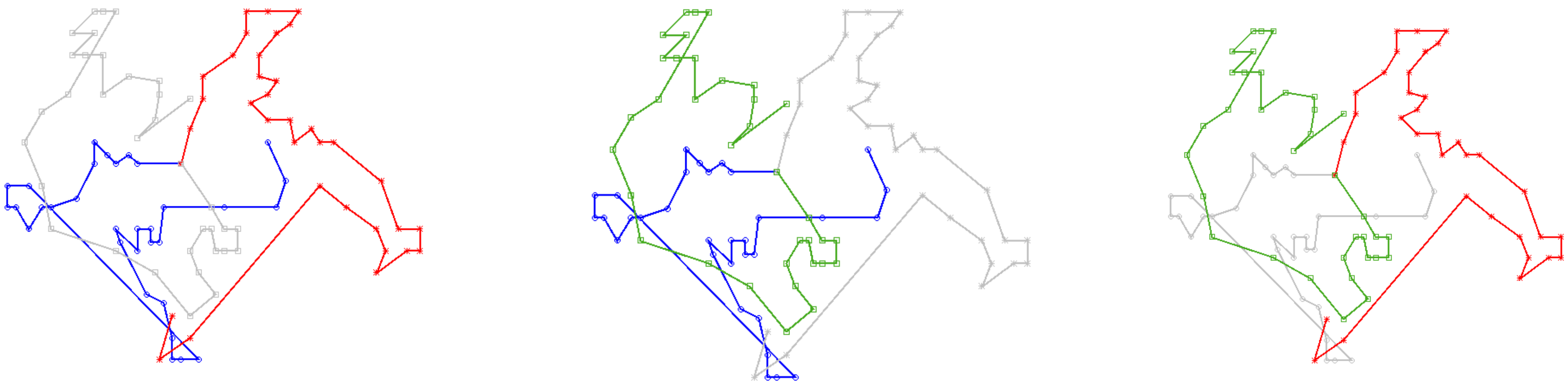
A MiniZinc VRP model

Advanced techniques – Recreate

Adaptive Decomposition

Decompose problem

- Only consider 2-3 routes
- Smaller problem is much easier to solve



Advanced techniques – Recreate

Adaptive Decomposition

Decompose problem

- Only consider 2-3 routes
- Smaller problem is much easier to solve

Adaptive

- Decompose in different ways
- Use problem features to determine decomposition

Conclusions

- Now you know
  - o How to construct a solution to a VRP by successive insertion
  - o How to improve the solution using
    - – Variable Neighbourhood Search
    - – Large Neighbourhood Search

- Argued that CP is “natural” for solving vehicle routing problems
  - – Real problems often have unique constraints
  - – Easy to change CP model to include new constraints
  - – New constraints don’t change core solve method
  - – Infrastructure for complete (completish) search in subproblems

- LNS is “natural” for CP
  - – Insertion leverages propagation