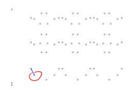
Construction algorithms

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

TSP

- "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 ← REALLY GOOD
  - Scheduling: Schedule the item with the earliest deadline ← OK

Construction algorithms – Minimum Insert Cost

A better greedy algorithm

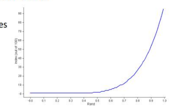
- "Minimum Insert" is another greedy algorithm – but performs slightly better
- At each iteration (OPTION TSP)
  - Find the customer that, when inserted, increases the cost by the least.
  - Insert it in the position that increases the cost by the least
  - Allows insert between *any* pair of customers – not just at the end

Construction algorithms – 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

- Combine "greedy" and "random"
- Don't always make the very "greediest" choice
- E.g. choose the r^{th} best of n choices
- $r = n * (\text{uniform}(0,1))^k$



Regret

i-th

regret(i) = C(insert i in 2^{nd}-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

- 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

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

LOOKAHEAD

Next time

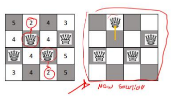
- Now we have our first solution … can we improve it?

FRIDAY QUIZ

n-queens

NO QUEUE — MAX FOOD — ON Ai: GREEDY BEST FIRST SEARCH (GBFS) QUEUE

- Alg 2: Greedy (a.k.a Hill Climbing)
  - Choose the best move / one of the best moves
  - Requires us to evaluate the entire Neighbourhood

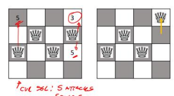


Hill Climbing has been compared to climbing Mt Everest in thick fog … and while suffering from amnesia.

n-queens

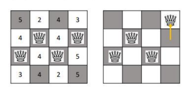
- Alg 3: First found
  - Randomise neighbourhood evaluation
  - Make first improving move

Computational evidence favours first-found over greedy



n-queens

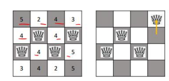
- Greedy search is incomplete
- Alg 4: Randomised Greedy
  - With probability p do greedy/first found move
  - With otherwise do a random move
    - Asymptotically complete



n-queens

MIN PROBLEM

- Alg 5: Biased
  - Choose a move with probability (inversely) proportional to score
  - (Twice as likely to choose a '2' move than a '4')



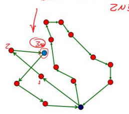
VRP

Local Search in VRP

- More complex operators
  - 1-move

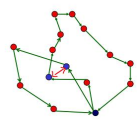
MOVE FROM 3RD TO 2ND

→ POS IN WHICH WE VISIT THE CLIENT



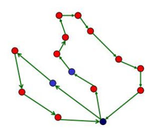
Other Neighbourhoods for VRP:

- Swap 1-1



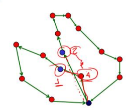
Other Neighbourhoods for VRP:

- Swap 1-1



Other Neighbourhoods for VRP:

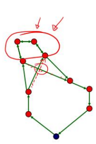
- Swap tails



VRP specific operators

2-opt (3-opt, 4-opt…)

- Remove 2 arcs
- Replace with 2 others

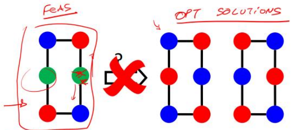


Neighbourhood

- E.g. Map Colouring (k-colouring)
  - Colour a map (graph) so that no two adjacent countries (nodes) are the same colour
  - Either:
    - Use at most k colours
    - Minimize number of colours

Map Colouring

FEAS — OPT SOLUTIONS



Starting sol — Two optimal solutions

Define neighbourhood as:

Change the colour of at most one vertex

Make k-colour constraint soft fixes this issue

Neighbourhood

- Hard constraints create impenetrable mountain ranges in solution space
- Soft constraints allow passes through the mountains

Strongly connected: MAP W/ 2 WAY STREETS ONLY

- Any solution can be reached from any other (e.g. 2-opt)

Weakly optimally connected MAP WITH SOME ONE-WAY STREETS

- The optimum can be reached from any starting solution

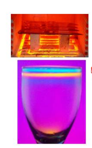
Escaping Plateaux (Shoulders)

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Need to place a limit on the possible number of sideways moves to avoid infinite loops
- For 8-queens
  - Now allow sideways moves with a limit of 100
  - Raises percentage of problem instances solved from 14% to 94%
- However….
  - 21 steps on average for every successful solution
  - 64 for each failure 6%

Escaping local minima II

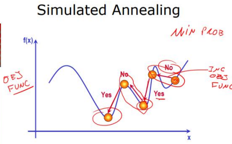
Solution 2: Simulated Annealing

- Based on manner in which crystals are formed
  - At high temperatures, molecules move freely
  - At low temperatures, molecules are "stuck"
- If cooling is slow
  - A low energy, organized crystal lattice formed
- Minimise energy in crystal ⟺ Minimise objective



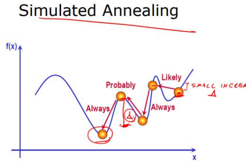
Simulated Annealing

MIN PROB



Greedy local search reduces objective at every iteration

Simulated Annealing



Simulated Annealing accepts/rejects solution candidate based on probability

Simulated Annealing

If candidate solution reduces the objective,

- always accept the change (c.f. first found)

If candidate solution has higher objective,

- system parameter T *("Temperature")* controls probability of acceptance

$$P(\text{accept increase of } \Delta \text{ in objective}) = e^{-\Delta/T}$$

- T reduces as method proceeds
- As $T \to 0$, only improving moves accepted

Meta-heuristics: Properties (1)

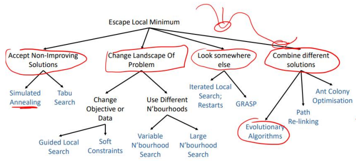
- can address both discrete- and continuous-domain optimisation problems
- are strategies that "guide" the search process (LOCAL S.)
- range from simple local search procedures to complex adaptive learning processes
- efficiently explore the search space to find good (near-optimal) feasible solutions
- provide no guarantee of global or local optimality B&B
- are agnostic to the unexplored feasible space (i.e., no "bound" information)
- lack a metric of "goodness" of solution (often stop due to an external time or iteration limit)

Meta-heuristics: Properties (2)

- are not based on some algebraic model (unlike exact methods)
- can be used in conjunction with an exact method
  - E.g., use meta-heuristic to provide upper bounds ← FEAS SOL
  - E.g., use restricted MILP as "local heuristic" (== matheuristic)
- are usually non-deterministic
- are not problem specific (but their subordinate heuristics can be)
- may use some form of memory to better guide the search

VRP 3-OPTs

Meta-heuristics: An Incomplete Survey



Meta-heuristics: Overview

EXPLOITATION VS EXPLORATION

Exhibit Intensification and Diversification
- Need to do both
- Can be explicitly controlled

Intensification (Exploitation):
- Concentrate search around already-found "good" solutions
- Look harder in a smaller area

Diversification (Exploration)
- Expand the area being looked at
- Find new (promising?) areas to search
- Includes mechanisms to avoid getting trapped in confined areas of the search space

Tabu Search

- **Taboo** (*English*): prohibited, disallowed, forbidden
- **Tabu** (*Fijian*): forbidden to use due to being sacred and/or of supernatural powers

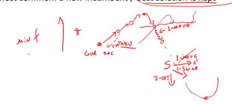
TABU

- Starts with classic Local Search until a local minimum is found
- Choose an objective-increasing move
- Make undoing that move "tabu"
  - Place it on a "tabu list"
- Repeat

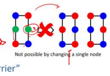
```
s ← GenerateInitialSolution()
TabuList ← ∅
while termination conditions not met do
  s ← ChooseBestOf(N(s) \ TabuList)
  Update(TabuList)
endwhile
```

Tabu Search

"Aspiration Criteria"
- Allowed to keep a solution if it meets certain criteria
- Most common: a new incumbent / best solution is kept



Soft Constraints



Not possible by changing a single node

General idea: Move constraints into the objective
- Relax Constraints
  - Penalise degree of violation
    - Allows Local Search to move through "infeasibility barrier"
  - Opens new areas of search space
  - Maintain both best feasible solution, "best" infeasible solution
  - Increase penalty over time to force incumbent back to feasibility
  - Often used with Simulated Annealing or Tabu Search
  - Extensively used in practice, e.g., VRP
- Parameters:
  - Which constraints to relax
  - Penalty schedule

Guided Local Search

LONG ARCS FOR VRP or TSP

Basic idea:
- Select a *feature* that indicates a poor solution
- Penalise a solution that exhibits that feature
- Start with zero penalty
- Repeat
  - Perform Local Search to minimise original objective **+ penalties**
  - Select elements to penalise
  - Increase penalty on selected elements

Guided Local Search

Local Search
- Do local search with an *updated objective* $h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s)$

$ MILLIONS

10 ~ 100

- Updated objective
  - h(s): Augmented objective
  - g(s): Original objective
  - λ: "normalisation" parameter
  - p_i: **Count** of times feature *i* has been penalised
  - $I_i(s)$: Indicator function: 1 if feature *i* in solution s; 0 otherwise

Guided Local Search

GLS

MOST BIG IN CUR SOL

ARC LENGTH

Select elements to penalise:
- Update penalty of features that maximise Utility $util(s_*, f_i) = I_i(s_*) \cdot \frac{c_i}{1 + p_i}$
- c_i: Original cost of feature
- $I_i(s)$: Does solution s exhibit feature *i*

CUR SOL

- At each iteration
  - Local Search using augmented objective $h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s)$
  - Select maximum utility features
  - Set $p_i = p_i + 1$ for all selected features
- Penalty increases each iteration
  - Local Search tries harder to eliminate feature
- Utility decreases the more often you penalise a feature
  - Eventually select other features

Guided Local Search – TSP

- Feature: an arc (*i,j*)
- $I_{ij}(s) = 1$ if arc (i,j) is present; 0 otherwise

$$h(s) = g(s) + \lambda \cdot \sum_{i=1}^{M} p_i \cdot I_i(s)$$

- At each iteration
  - Penalise the longest arc.
  - Try harder to get rid of it

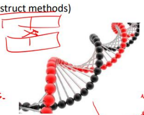
$$util(s_*, f_i) = I_i(s_*) \cdot \frac{c_i}{1 + p_i}$$

- As the search progresses
  - Utility of first arc decreases
  - Other arcs start being penalised

Genetic Algorithms

- Generate a population of solutions (construct methods)
- Evaluate fitness (objective)
- Create next generation:
  - Choose two solutions from population
  - Combine the two (two ways)
  - [Mutate]
  - Produce offspring (calculate fitness)
  - [Improve] LOCAL SEARCH ON OFFS.
  - Repeat until population doubles
- Apply selection:
  - Bottom half "dies"
- Repeat

Turns it into a *Memetic Algorithm*



NEW OFFSP.

Solution Representation is key
- Needs to fulfil **multiple goals**
  - Easy to calculate fitness (objective)
  - Easy to perform crossover (merge)
  - Easy to manipulate (mutation)
  - Easy for local search
- E.g. VRP
  - First attempts used array for each route, or successor info
  - Very difficult for crossover
  - Better rep turns out to be a single array

D → 3 → 2 → 3 → D

D → 3 → 2 → 3 → D S_1

D → 3 → 3 → D S_2

3 2 1

BAD

Diversification:
- Big problem is getting a homogenous population
- Too much intensification, not enough diversification

EXPLORATION

- Some algorithms explicitly measure diversity
  - keep lower-quality solutions that maintain diversity
- Meta-meta Soft constraints
  - Maintain a separate population of infeasible solutions

Ant Colony Optimisation (ACO)

TCP PACKETS IN NET COLL. COM.

E.g.: Want to find shortest paths in a communications network
- Send out ants that choose a path
  - Partly at random (our naive heuristic)
  - Partly influenced by previous ants: **Pheromone trail** – our meta-heuristic
- The first ant to get to a destination increases the Pheromone on its path
- Pheromone levels decrease over time
  - More ants select the best path
  - The best path gets reinforced

(ACO) – TSP

Probability that ant chooses arc *xy*
- τ : Pheromone strength
- η : A–priori "attractiveness"

$$p^k_{xy} = \frac{(\tau^{\alpha}_{xy})(\eta^{\beta}_{xy})}{\sum_{z \in \mathrm{allowed}_x} (\tau^{\alpha}_{xz})(\eta^{\beta}_{xz})}$$

Path Relinking

Basic idea:
- Take two solutions
- "Walk" between them

For TSP:

GA POP-BASED ALG BIG # OF OFFS

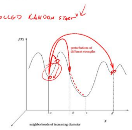
4 DIF SOL.

S_1	1	4	6	2	3	8	9	5	10	7
	1	10	6	2	3	8	9	5	4	7
	1	10	8	2	3	6	9	5	4	7
	1	10	8	2	5	6	9	3	4	7
	1	10	8	2	5	4	9	3	6	7
S_2	1	10	8	2	5	4	9	3	6	7

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



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)

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 $O(N^3)$

$\mathcal{N}_i \not\subseteq \mathcal{N}_{i-1}$

$\mathcal{N}_i \cap \mathcal{N}_{i-1} \neq \emptyset$

NO REQ

GRASP

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

VRP

≠ ML FEAT.

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

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 NGHBH

LOCAL MIN MIGHT CHANGE

Very successful method across multiple domains

```
S = initialSolution()
k = 0
while k < m
  if LocalSearch(S, N_k) improved the solution S:
    k = 0
  else
    k = k + 1
```

1 — m

0 — m-1

Large Neighbourhood Search

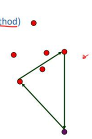
- Originally developed by Paul Shaw (1997)
- This version Ropke & Pisinger (2007)[1]
- A.k.a "Record-to-record" search

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

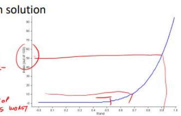
Large Neighbourhood Search

Destroy part of the solution (Select method)
- Examples
- Remove a sequence of visits



Destroy part of the solution (Select method)

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



TOP 60% 5 WORST

Destroy part of the solution (Select method)

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

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

Load

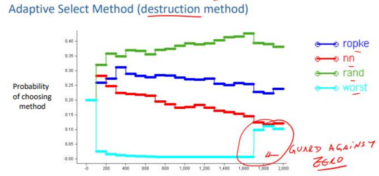
Distance

Time

Large Neighbourhood Search

→ SIZE

Adaptive Select Method (destruction method)



GUARD AGAINST ZERO

Large Neighbourhood Search

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

WE DON'T NEED OPT

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

Adaptive Insert Method ← RECONSTR.
- Ropke[1] adapts choice based on prior performance
  - "Good" methods are chosen more often

WHEN A METHOD FINDS A BETTER SOL → GIVE REAL TINY REW

START W/ UNIF RAND