Meta-heuristic Algorithms

Dr Rong Qu, Associate Professor
ASAP Group, The University of Nottingham
rong.qu@nottingham.ac.uk
http://www.cs.nott.ac.uk/~pszrq

NATCOR – Heuristics and Approximate Algorithms
Nottingham, April, 2016

Optimisation Problems

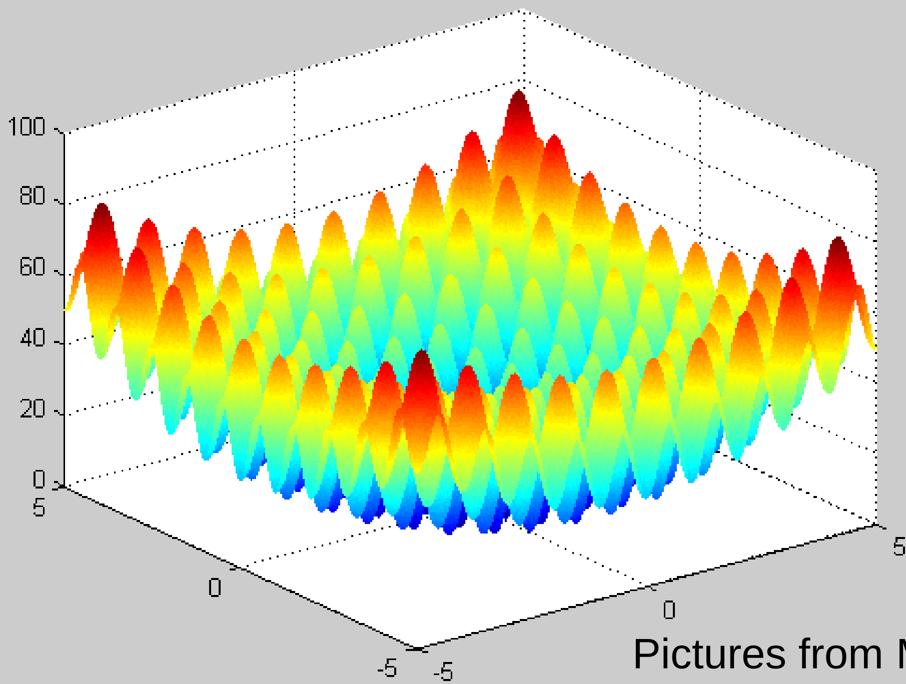
- For a set of decision variables: $X = (x_1, x_2, \ldots, x_n)$
  Maximises (or minimises) an objective function: $f(X)$
  Subject to a set of constraints

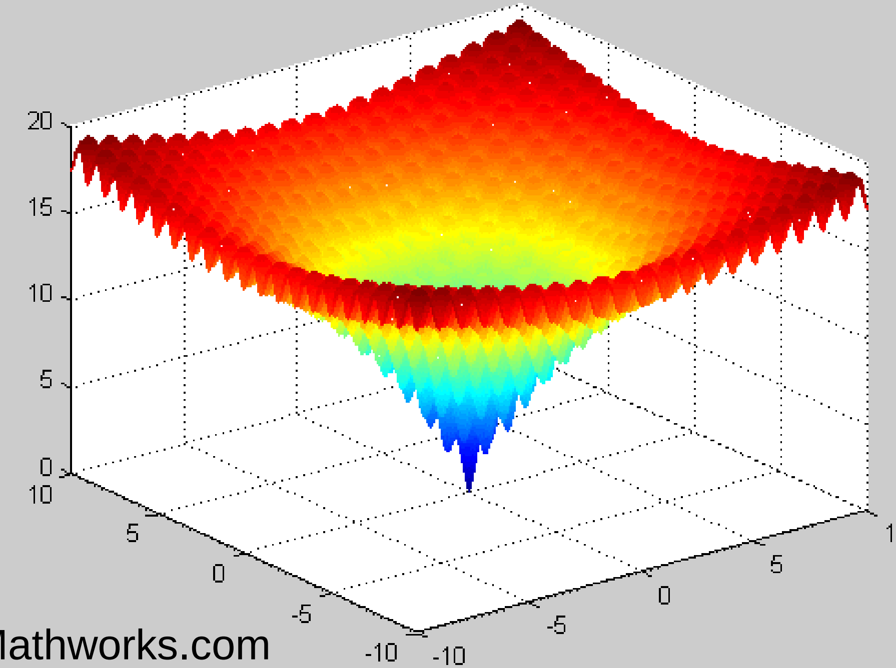
$$Ras(x) = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2).$$

$$x_i \in [-5.12, 5.12]$$

$$F(\vec{x}) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)\right) + 20 + e$$

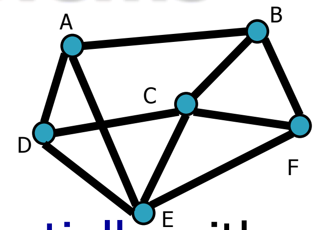
$$x_i \in [-30, 30]$$



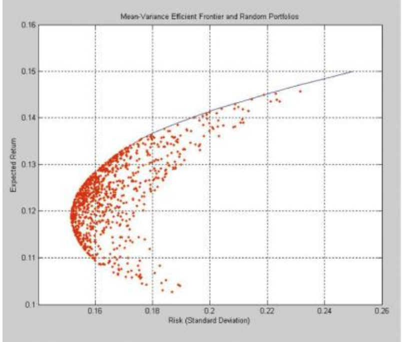


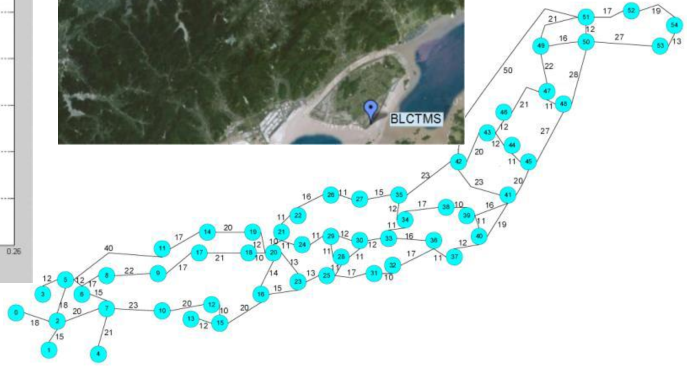
Pictures from Mathworks.com

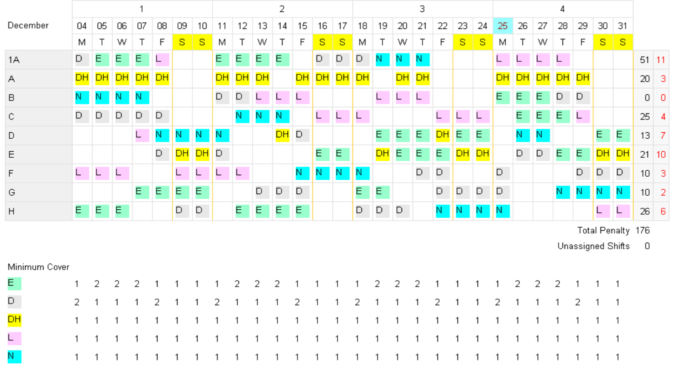
Combinatorial Optimisation Problems

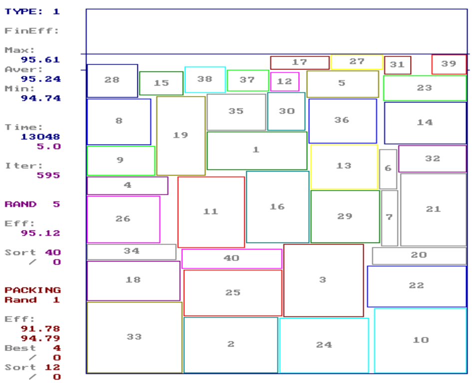


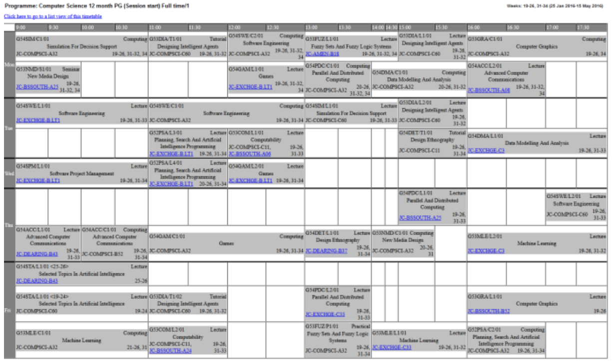
- For most of real world optimisation problems
  - An exact model cannot be built easily
  - Combinatorial explosion: no. of solutions grows exponentially with the size of the problem
- Search algorithms
  - Exact methods: IP, MIP
  - Constructive heuristics
  - Meta-heuristic algorithms





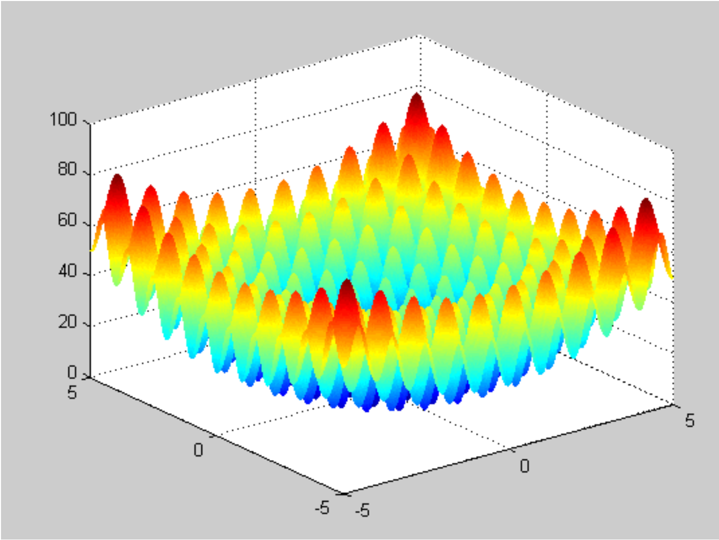






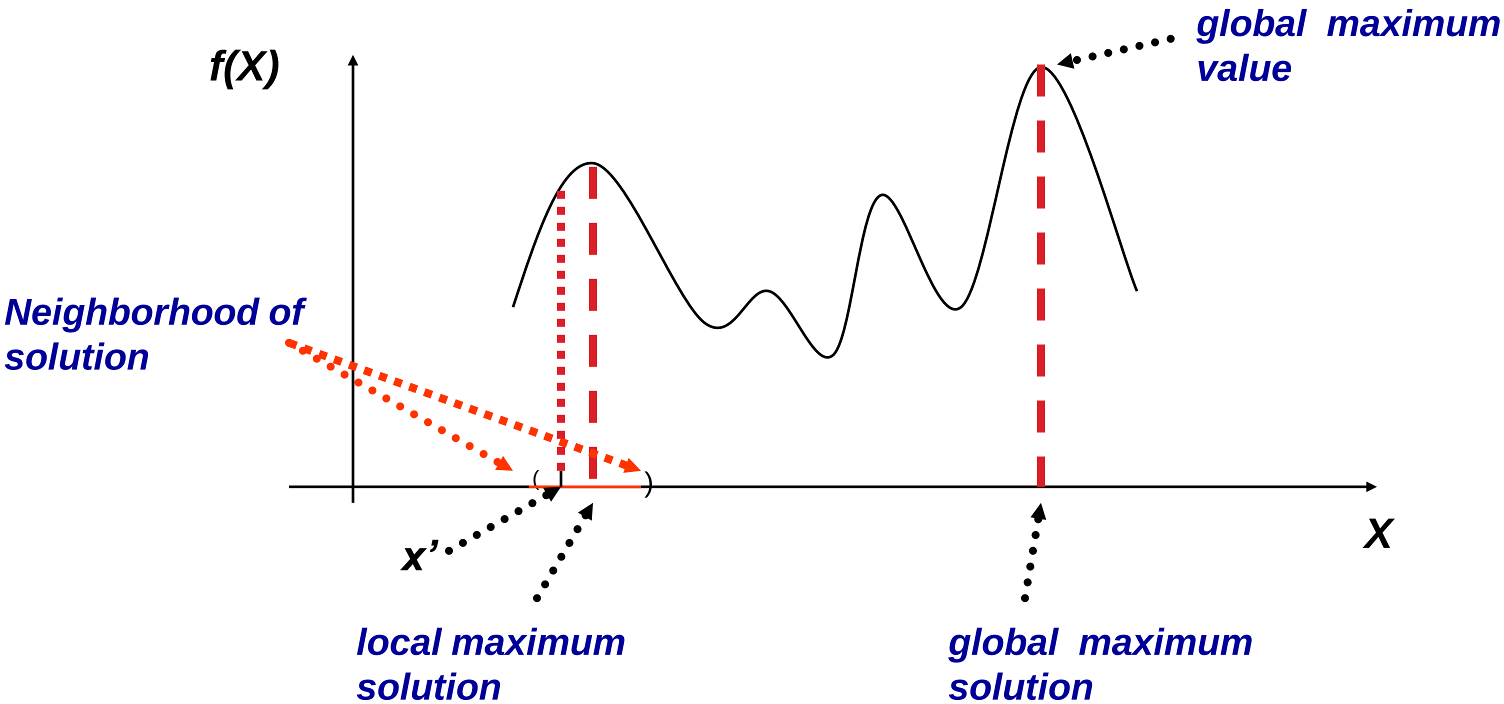
Combinatorial Optimisation Problems

- Constructive Heuristics
  - Simple minded greedy functions: iteratively build a reasonable solution, one element at a time
- Meta-heuristics
  - Single solution based (local search)
    - Simulated Annealing, Tabu Search, Variable Neighbourhood Search, etc.
  - Population based
    - Genetic algorithm, Memetic algorithm, EDA, Ant Algorithms, Swarm Intelligence, etc.

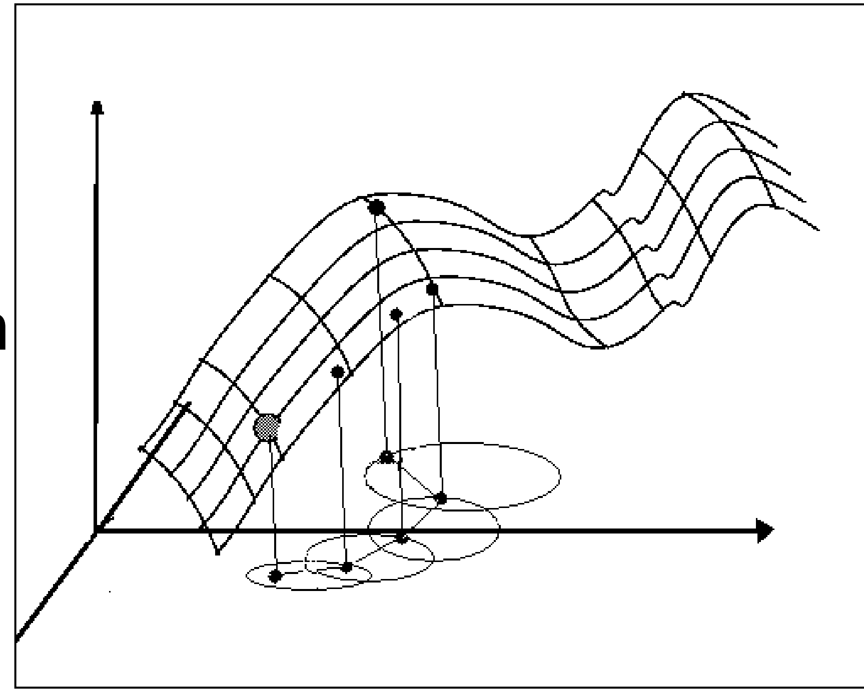


Local Search

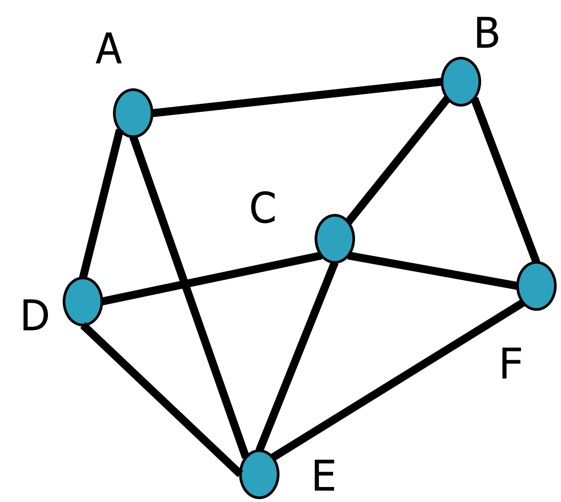
- Starts from initial (complete) solution
- Iteratively moves to a better neighborhood solution until a local optimum (no better neighborhood)



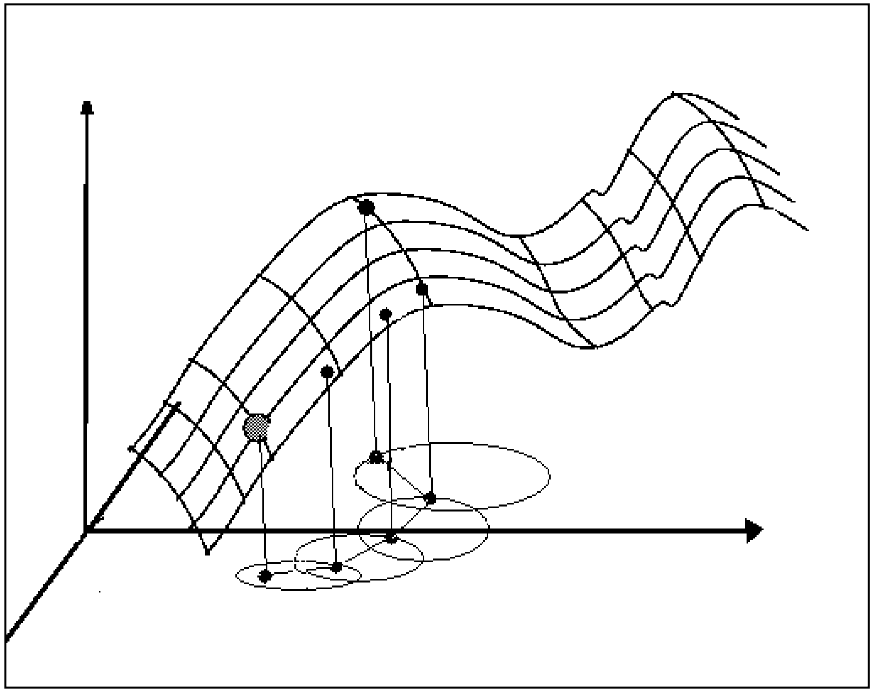
Local Search

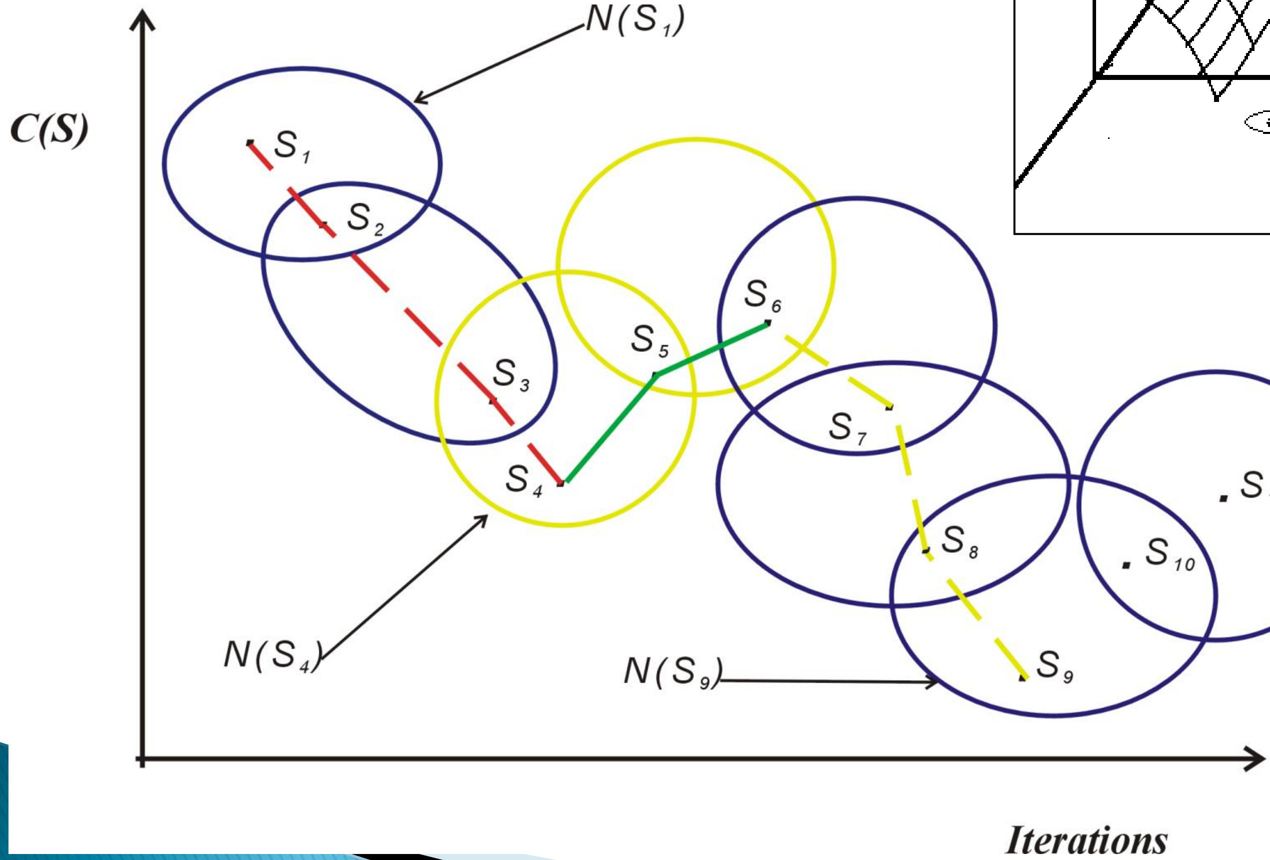


- Representation of the solution
  - Solution encoding
- Evaluation function
  - Guide the search
- Neighbourhood function
  - An operator to change (move) a solution to other solutions
- Acceptance criterion
  - First improvement, best improvement, best of non-improving solutions

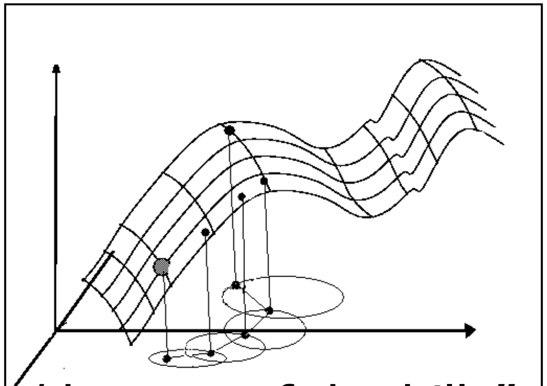


Local Search





Local Search



- Hill climbing / Steepest Descent
  - "Run uphill / downhill and hope you find the top / bottom of the hills"
- Simulated annealing
  - "Shake it up a lot and then slowly let it settle"
- Tabu search
  - "Don't look under the same lamp-post twice"
- Variable Neighbourhood Search
  - "Let's use different transportations i.e. fly / leap / walk, to explore"
- Etc.

- Population based approach
  - Genetic algorithms: "survival of the fittest"
  - Ant algorithms: "wander around a lot and leave a trail"
  - Genetic programming: Learn to program
  - Etc.

Simulated Annealing

- Physical annealing process: Material is heated and slowly cooled into a uniform structure

- The first SA algorithm: (Metropolis, 1953)
- SA applied to optimisation problems: (Kirkpatrick, 1982)

- Better moves are always accepted
- Worse moves may be accepted, depends on a probability

Kirkpatrick, S , Gelatt, C.D., Vecchi, M.P. 1983. Optimization by Simulated Annealing. Science, 220(4598): 671-680.

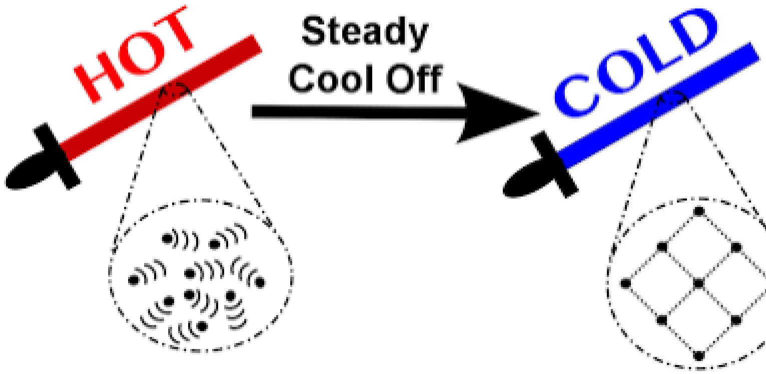


Figure 1: Sword Annealing Analogy to Explain Simulated Annealing (Copyright Jonathan Becker)

Simulated Annealing

- At temperature *t*, the probability of accepting a worse solution:

$$P = \exp^{(-|c|/t)} > r$$

  - *c* : change in the evaluation function
  - *r* : a random number between 0 and 1
  - *t* : the current temperature

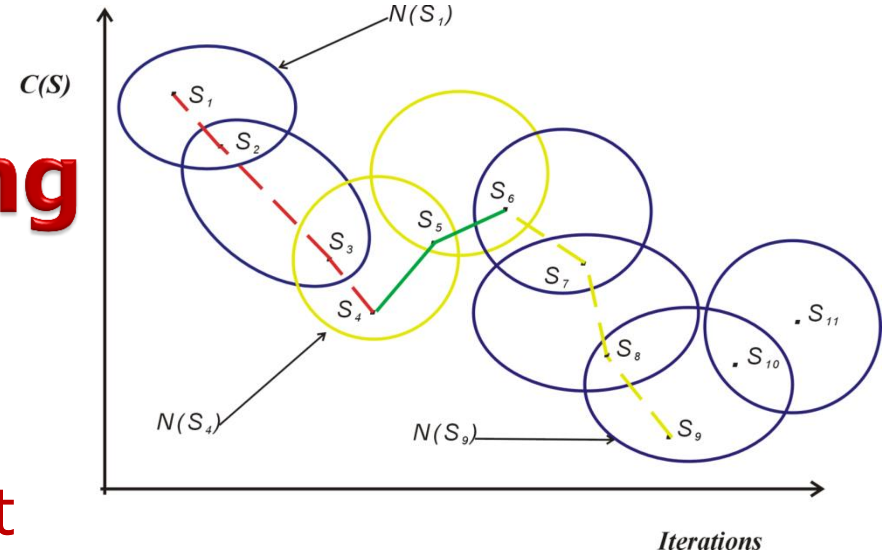
- The probability of accepting a worse state is a function of
  - the temperature t of the system
  - the change c in the cost function

Simulated Annealing

- The probability of accepting a worse state is a function of
  - the temperature t of the system
  - the change c in the cost function
- t decreases: the probability of accepting worse moves decreases
- t = 0: no worse moves are accepted (i.e. greedy search)

Change	Temp	$exp^{(-C/T)}$	Change	Temp	$exp^{(-C/T)}$
0.2	0.95	0.810	0.2	0.1	0.13583
0.4	0.95	0.656	0.4	0.1	0.018339
0.6	0.95	0.532	0.6	0.1	0.0024852
0.8	0.95	0.431	0.8	0.1	0.000335

Simulated Annealing



For I = 1 **to** Iter **do**
t = Schedule[I]
If t = 0 **then return** Current
Next = random neighbour of Current
c = evaluate[Next] – evaluate[Current]
if c > 0 **then** Current = Next
else Current = Next with probability $\exp^{(-|c|/t)}$

- Implement SA : implement greedy search + modified acceptance criteria $\exp^{-|c|/t}$
- Cooling Schedule is *hidden* in this algorithm: important!

SA – Cooling Schedule

- Starting Temperature
- Final Temperature
- Temperature Decrement
- Iterations at each temperature

SA – Cooling Schedule

- Starting Temperature
  - *hot* enough: to allow *almost* all neighbourhood (else: greedy search)
  - *not* be so hot: random search for sometime
  - Estimate a suitable starting temperature:
    - Reduce quickly to 60% of worse moves are accepted
    - Use this as the starting temperature

- Final Temperature
  - Usually 0, however in practise, not necessary
  - t is low: accepting a worse move are almost the same as t = 0
  - The stopping criteria: either be a suitably low t, or "frozen" at the current t (i.e. no worse moves are being accepted)

SA – Cooling Schedule

- Temperature Decrement
  - Enough iterations at each t, however computationally expensive
  - Compromise
    - Either: a large number of iterations at a few t's, or
    - A small number of iterations at many t's, or
    - A balance between the two
  - Linear: $t = t - x$
  - Geometric: $t = t * a$
    - Experience: α = (0.8 and 0.99)
    - The higher the value of α, the longer it will take

SA – Cooling Schedule

- Iterations at each temperature
  - A constant number of iterations at each t, or
  - One iteration at each t, but decrease t *very* slowly (Lundy 1986)
    - $t = t / (1 + \beta t)$
    - where β is a suitably small value
  - An alternative: dynamically change the no. of iterations
    - At higher t's: less no. of iterations
    - At lower t's: a large no. of iterations, local optimum fully exploited

SA – Acceptance $\exp^{(-|c|/t)}$

$\exp^{(-|c|/t)}$: took about one third of the computation time

- Approximates the exponential (Johnson, 1991)

$$P(c) = 1 - |c|/t$$

- Build a look-up table: values of $|c|/t$
- Speed up the algorithm: about a third with no significant effect on solution quality

Tabu Search

"The overall approach is to avoid entrapment in cycles by forbidding or penalizing moves … in the next iteration to points in the solution space previously visited (hence *tabu*)."

Proposed independently by Glover (1986) and Hansen (1986)

- Accept the best one, even it's low quality (worse move)
- Accepts worse solutions deterministically, to escape from local optima

Glover, Fred W., Laguna, Manuel. Tabu Search, Springer, 1996

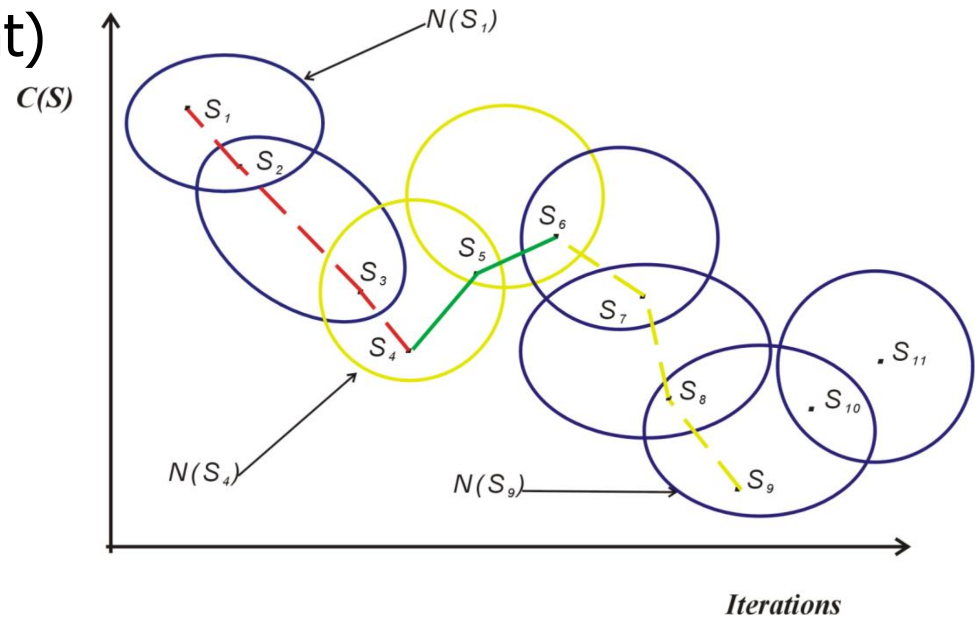
Tabu Search

- Uses memory (tabu list) to improve decision making
  - Short term memory: prevent revisiting previous solutions
    - Tabu list: Records a limited no. of solution attributes (moves, selections, assignments, etc.)
    - Tabu tenure (length of tabu list): No. of iterations a move is prevented
      - FIFO, dynamic
  - Long term memory: attributes of elite solutions
    - Diversification: Discouraging attributes of elite solutions, to diversify the search to other areas of solution space
    - Intensification: Give priority to attributes of a set of elite solutions

- Aspiration criteria: accepting an improving solution even it's generated by a tabu move
  - Similar to SA: always accepts better solutions, but accept worse ones

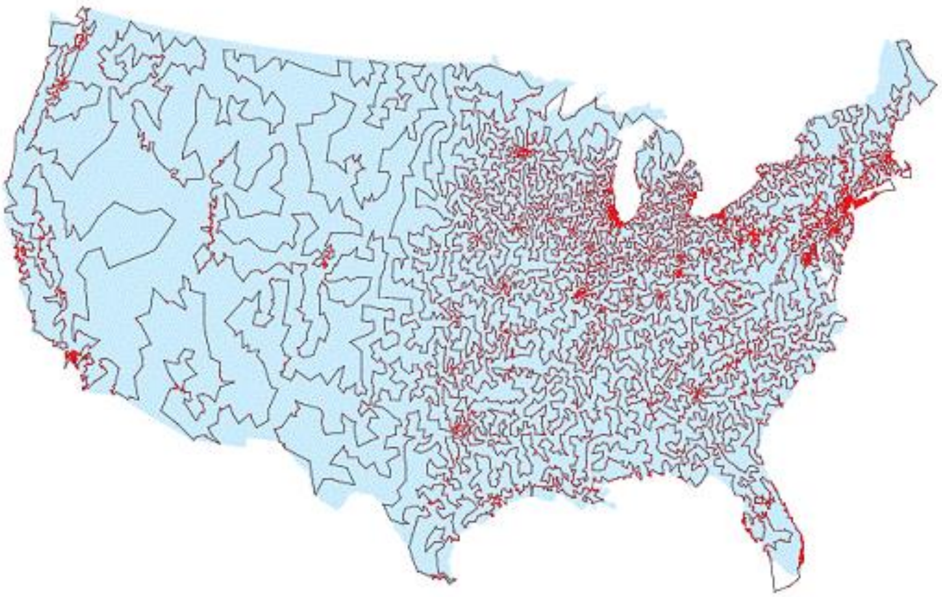
Tabu Search

Current = initial solution
While not terminate
  Next = the best neighbour of Current
  If(not MoveTabu(**TL**, Next) **or** Aspiration(Next)**) then**
    Current = Next
    Update BestSolutionSeen
    TL = Recency(**TL** + Current)
  Endif
End-While
Return BestSolutionSeen



Tabu Search – TSP Example

- Short term memory
  - Prevent a list of *t* towns from being selected for a no. of iterations
- Long term memory
  - Maintain a list of *t* towns in the last *k* best (worst) solutions
  - Encourage (or discourage) their selections in future solutions
- Aspiration
  - Moves in the tabu list generate better solution: accept that solution anyway
  - Put it into tabu list



Tabu Search vs. Simulated Annealing

	SA	TS
No. of neighbours at each move	1	n
Accept worse moves? How?	Yes by $P = \exp^{(-c/t)}$	Yes, the best neighbour if it is not tabu-ed
Accept better moves?	Always	Always (aspiration)
Stopping conditions	t = 0, or At a low temperature, or No improvement after some iterations	Certain no. of iterations, or No improvement after some iterations

Variable Neighbourhood Search

- In most local search: only one neighbourhood

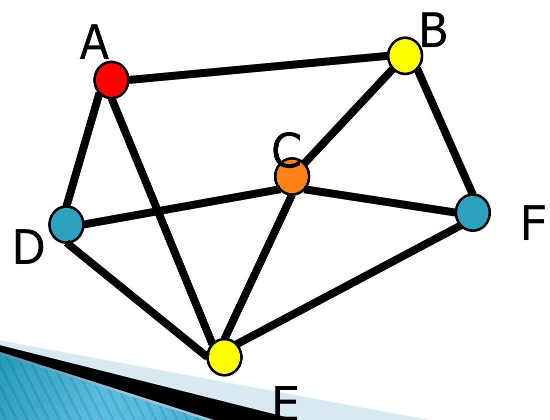
- To escape from local optimum
  - SA: move to worse neighbourhoods based on a probability using cooling schedule
  - TS: move to not tabued worse neighbourhoods

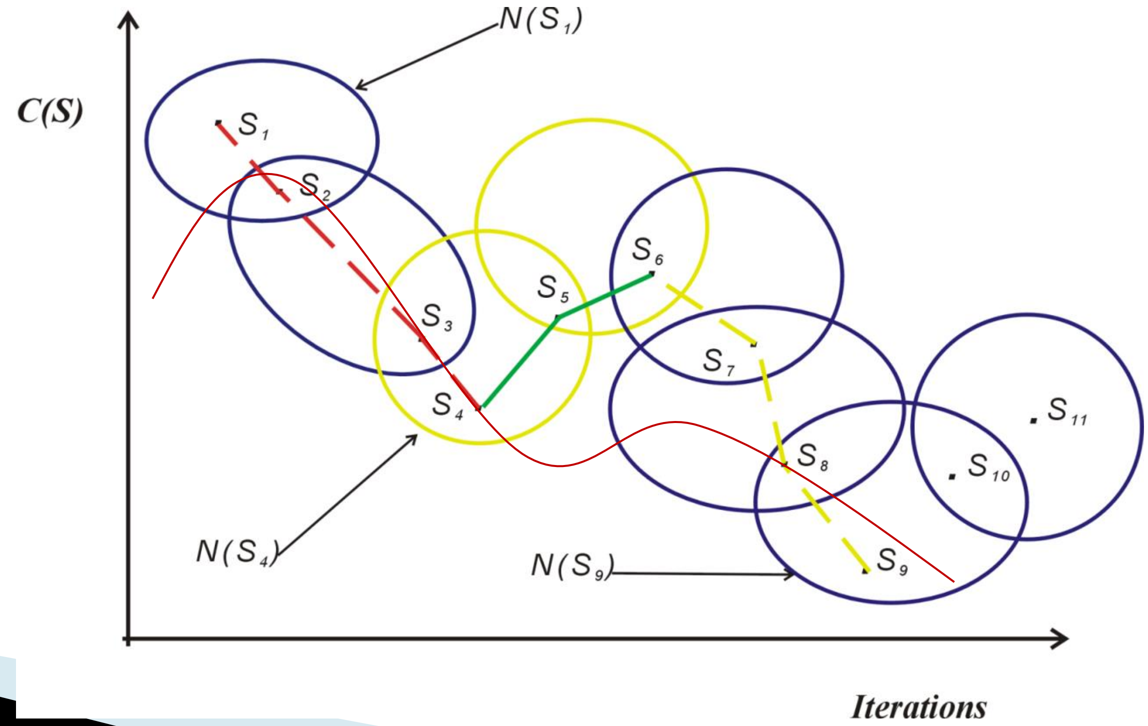
- **VNS:** systematically changes neighbourhood during search
  - N_k, $k = 1, 2, \ldots k_{max}$: the set of neighbourhood operators
  - $N_k(s)$: set of solutions in the k^{th} neighbourhood of solution s

P. Hansen and N. Mladenovic, Variable neighbourhood search: Principles and applications, EJOR 130: 449-467, 2001

Variable Neighbourhood Search

- **Fact 1.** A local minimum w.r.t. one neighbourhood is not necessary so for another
- **Fact 2.** A global minimum is a local minimum w.r.t. all possible neighbourhood
- **Fact 3.** For many problems local minima w.r.t several neighbourhoods are relatively close to each other



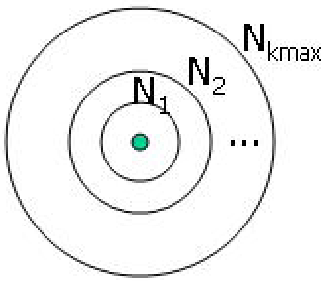


Variable Neighborhood Search

Initialisation

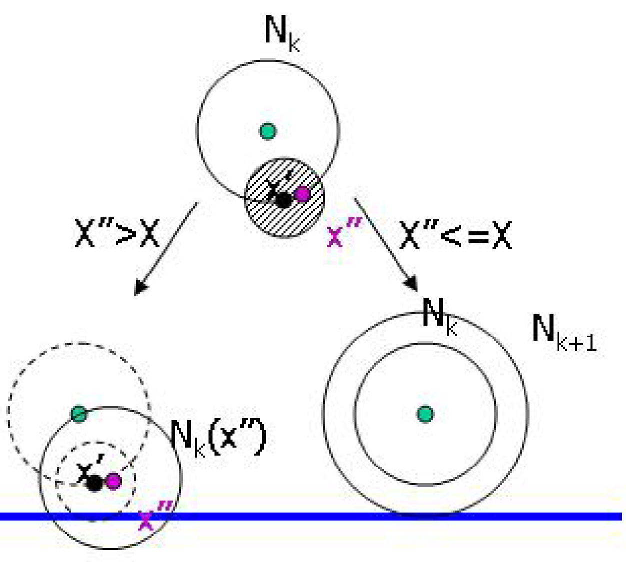
Select the set of neighbourhood structures N_k

Find an initial solution x



Repeat until stopping condition is met

- Set K=1
- **Repeat** until $k=k_{max}$
  1. *Shaking*: Generate a random point X′ in $N_k(x)$
  2. *Local Search*: x″ is the obtained optimum
  3. Move or not:
     - If x″ is better than x then x=x″ and k=1
     - Otherwise k=k+1



Talbi, **Metaheuristics – From design to implementation**, Wiley, 2009

Variable Neighbourhood Search

- Order of neighbourhoods
  - Typically, order neighbourhoods from smallest to largest
  - Forward VNS: start with $k = 1$ and increase k by one if no better solution is found; otherwise set $k \leftarrow 1$
  - Backward VNS: start with $k = k_{max}$ and decrease k by one if no better solution is found
  - Extended version: parameters k_{min} and k_{step}; set $k \leftarrow k_{min}$ and increase k by k_{step} if no better solution is found

Variants of VNS

Procedure **Reduced VNS**

Select $\{N_k\}$, k = 1, …,k_{max}, initial solution x, stopping condition

$k \leftarrow 1$

Repeat until $k = k_{max}$

$x' \leftarrow$ RandomSolution($N_k(x)$)

if $f(x') < f(x)$ then

$x \leftarrow x'$

$k \leftarrow 1$

else $k \leftarrow k + 1$

End

- Same as basic VNS except: no LocalSearch is applied
- Only explores randomly different neighbourhoods
- Can be faster than standard local search

Design VNS

- Number and type of neighbourhoods to be used
- Order of their use in the search
- Strategy for changing the neighbourhoods
- Local search methods
- Stopping condition

- No need of sophisticated acceptance criteria to escape from local optima
- Neighbourhoods: crucial for VNS; all solutions reachable!

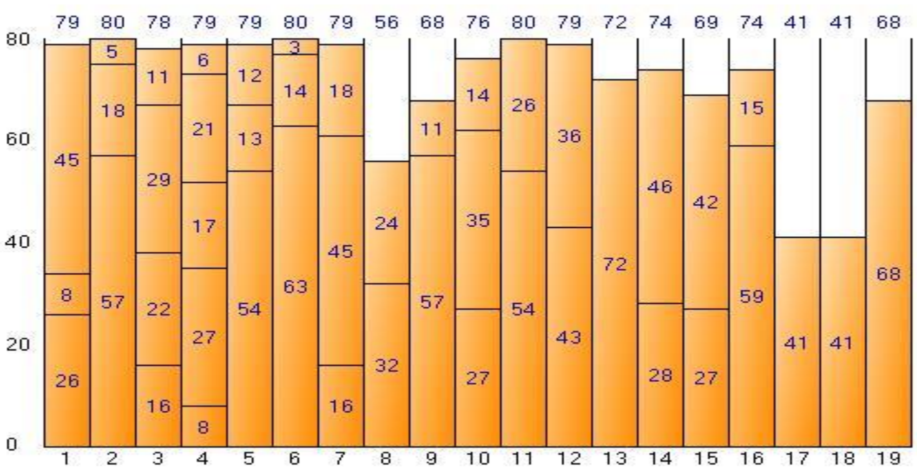
- Exercise: Design a VNS for TSP

Local Search: Design & Improve

- Evaluation function
  - Calculated at every iteration
  - Often the most expensive part of the algorithm
  - Need be as efficiently as possible
    - Delta / partial evaluation
    - Approximate evaluation function, potentially good solutions fully evaluated

Local Search: Design & Improve

- Evaluation function
  - If possible, should lead the search
    - Avoid where many states return the same value
      i.e. a plateau in the search space, the search has no knowledge where it should proceed



Local Search: Design & Improve

- Evaluation function
  - Cater for some illegal solutions using constraints
    - Hard Constraints :
      - cannot be violated in a feasible solution
      - a large weighting: these illegal solutions have a high cost
    - Soft Constraints :
      - should, ideally, not be violated but, if they are, the solution is still feasible
      - weighted depending importance
    - Can be dynamically changed as the algorithm progresses.
      - Allows hard constraints to be accepted at the start of the algorithm but rejected later

Local Search: Design & Improve

- Initial solution
  - A random solution: improve
  - A solution that's been heuristically built (e.g. for the TSP problem, start with a greedy search)

- Hybridisation
  - Combine two search algorithms
  - The primary search : a population based search
  - A local search is applied to move each individual to a local optimum

Other Local Search Metaheuristics

- Iterative Local Search
- Guided Local Search
- GRASP (Greedy Random Adaptive Search Procedure)
- And many more

- Software Tool
  - Andrea Schaerf, Marco Cadoli and Maurizio Lenzerini. LOCAL++: A C++ framework for local search algorithms. Software: Practice and Experience, 30(3): 233–257, 2000