CS [illegible]900

Se[illegible]ware

Engi[illegible]SE)

Erik [illegible]icks (fredericks@oakla[illegible])

Fall 2[illegible]19

CIS641
Search-Based Software Engineering (SBSE)

Erik Fredericks (frederer@gvsu.edu)

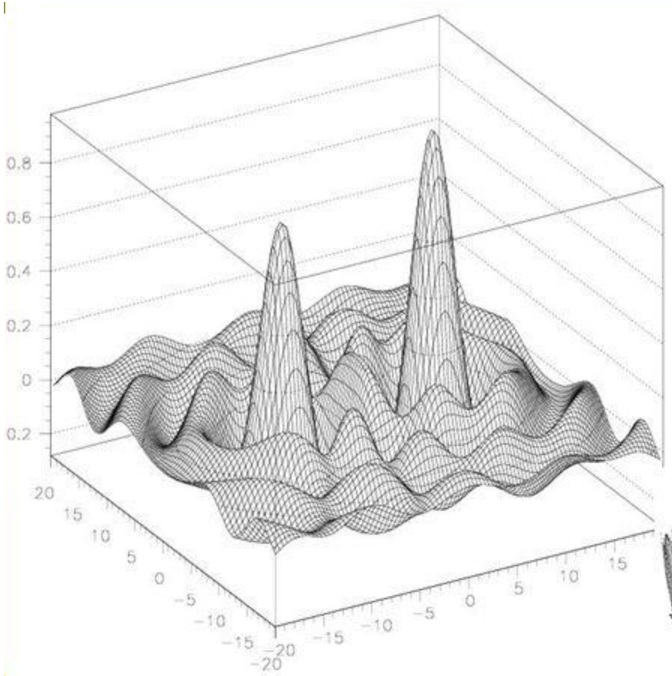
Overview

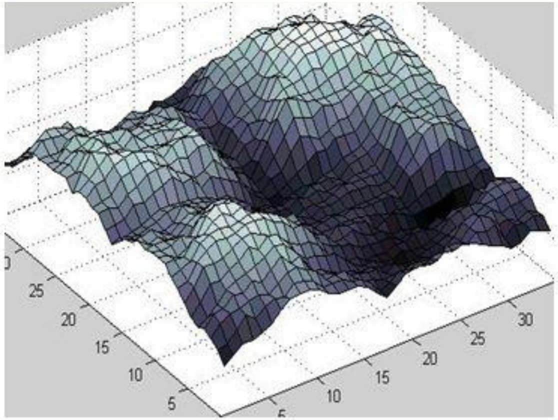
SBSE

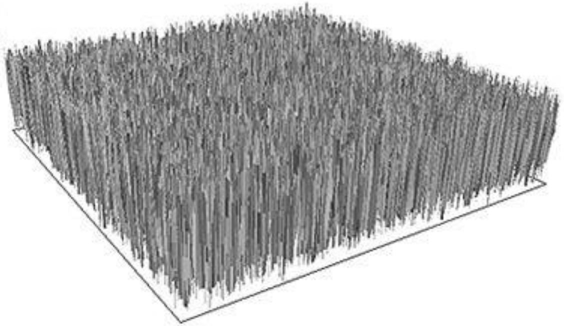
Search algorithms

Applications to SE and CPS (and ISE ha ha ha HAAA)

Basics of Search Algorithms

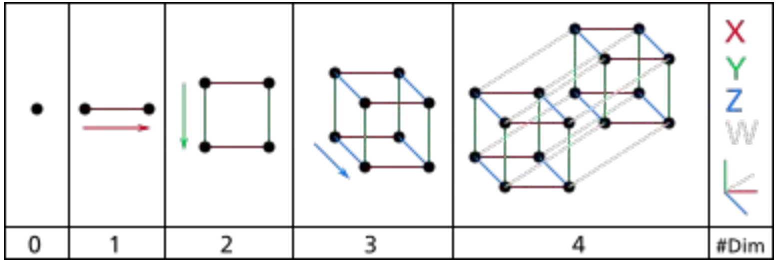


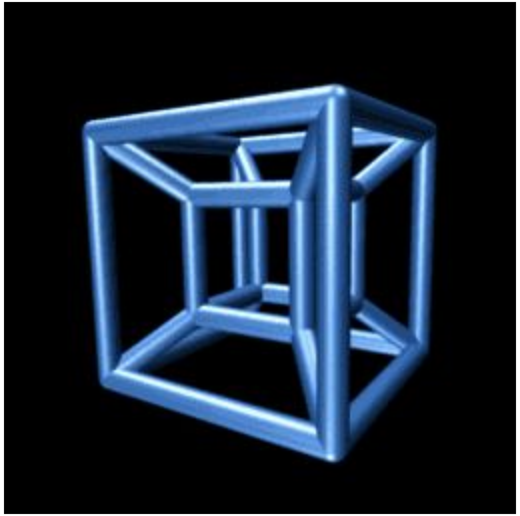




Basics of Search Algorithms

This is a tesseract -- a 4D representation of an object



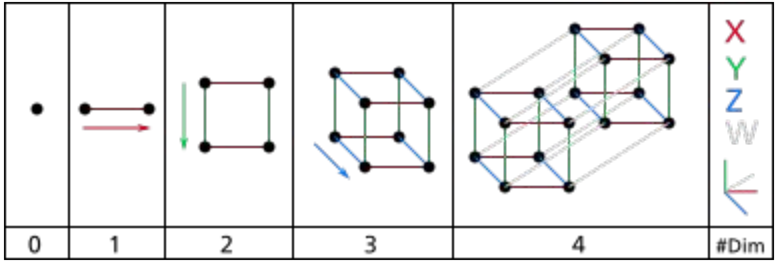


Why am I showing you this?

Basics of Search Algorithms

Relate it to SE

Dimension = *point of optimization*

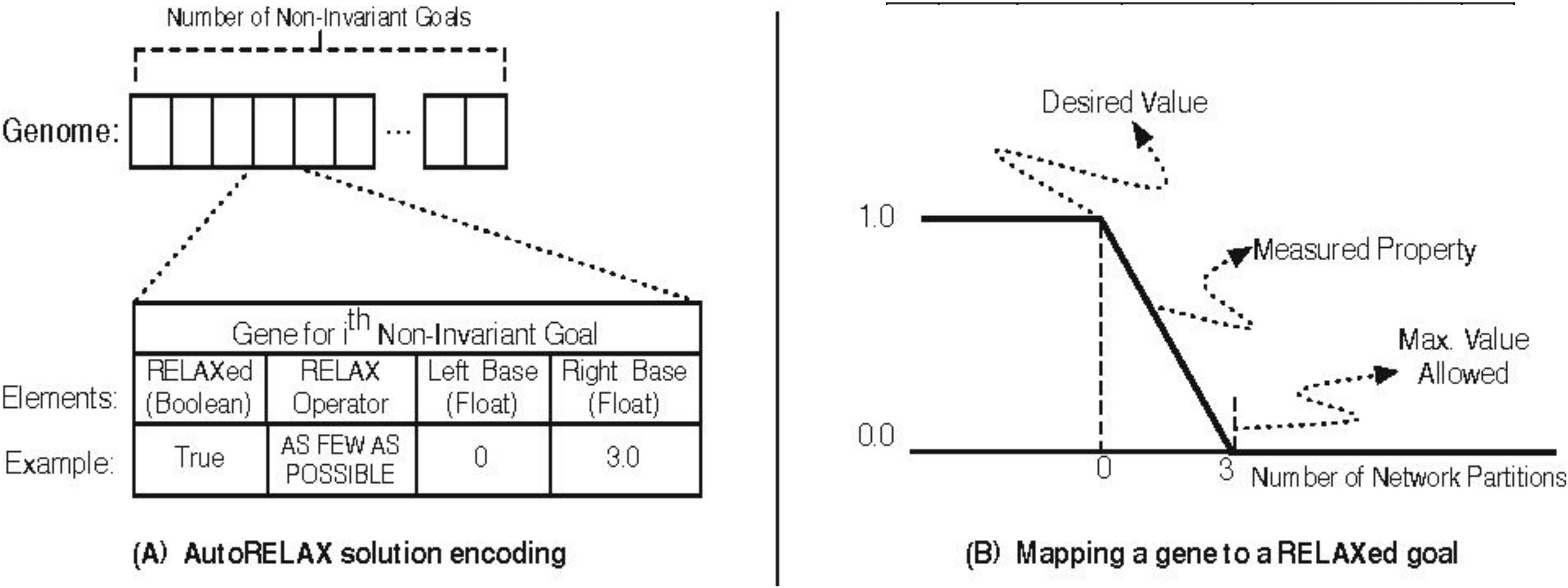


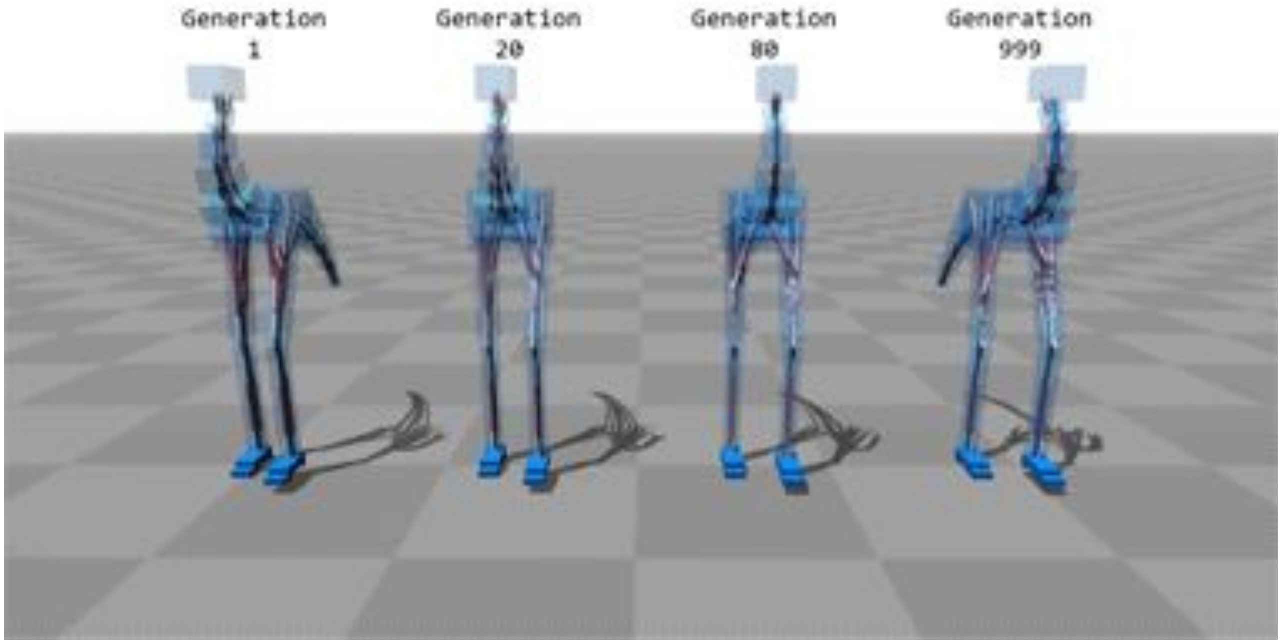
Consider a KAOS model and RELAX operators

1 dimension might be if a goal is RELAXed

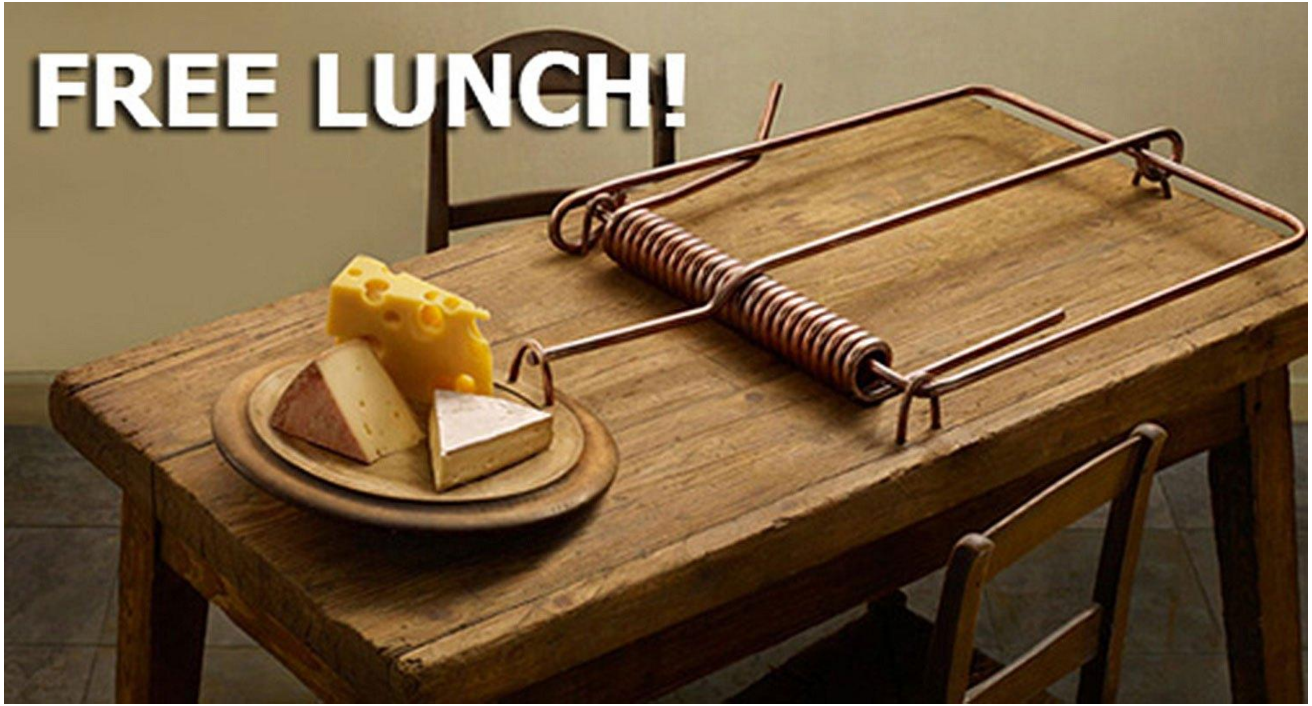
Another dimension might be which RELAX operator to apply

Basics of Search Algorithms




Generation
1
Generation
20
Generation
80
Generation
999

“No free lunch”



What are “search” problems for SE?

Think about all the software engineering *artifacts* we’ve talked about so far…

SBSE

Search-based software engineering

- Application of search-based techniques to software engineering problems

Examples:

- Automatically generating *optimal* test suites
  - Test case inputs
  - Ordering of test cases
  - Etc.
- Detecting incomplete requirements with symbolic analysis
- Optimizing self-adaptive system reconfigurations

What generally *comprises* a search heuristic

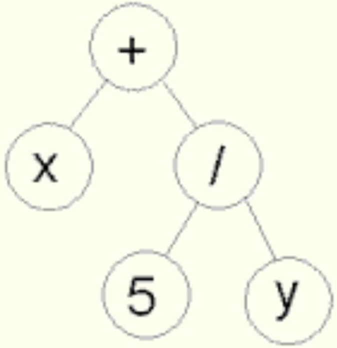
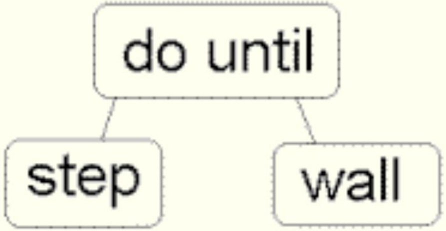
Representation

- How solutions are **encoded**
- Could be a binary number, vector (bounded or unbounded), a tree, etc.
  - A *data structure* that represents your configuration

Fitness function

- How solutions are **evaluated**
- **This is generally the hardest part of any search algorithm**

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do_until step wall)

http://www.obitko.com/tutorials/genetic-algorithms/encoding.php

Interestingly...

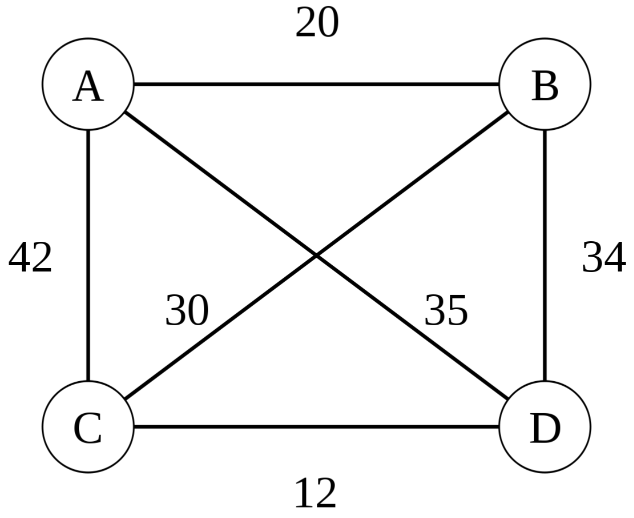
Search algorithms in general are **ridiculously easy to program**

- Most involve a for loop with some data structure manipulation / update functions

As was mentioned, simply **understanding your search space** is the hardest part

- Encoding valid solutions
- Calculating fitness / goodness / cost

Motivating Example



Traveling salesman problem (TSP)

- Minimize distance traveled between all cities
- Each city visited once
- Final city is start city

May be symmetric (*bi-directional paths*) or asymmetric (*uni-directional paths*)

NP-complete problem (NP and NP-hard)

- Easy to verify solution
- No easy way to figure out the ‘correct’ solution
- No guarantee for optimum in polynomial time

Great for testing out optimization algorithms

Motivating Example

Traveling salesman problem (TSP)

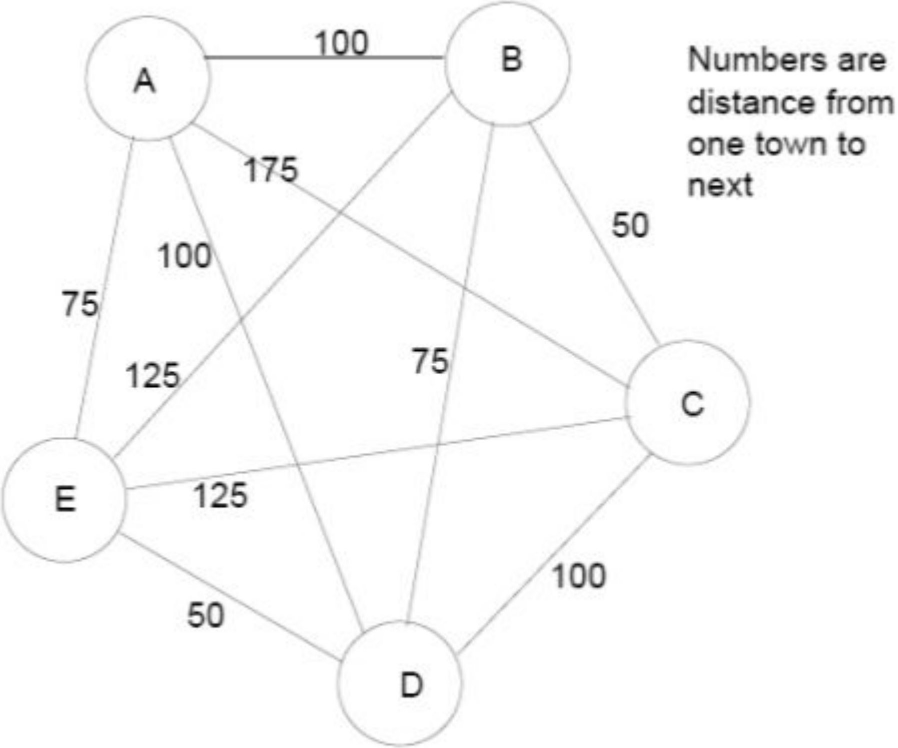
- Minimize distance traveled between all
- Each city visited once
- Final city is start city

May be symmetric (*bi-directional paths*) or a

NP-complete problem (NP and NP-hard)

- Easy to verify solution
- No easy way to figure out the ‘correct’ solution

Great for testing out optimization algorithms

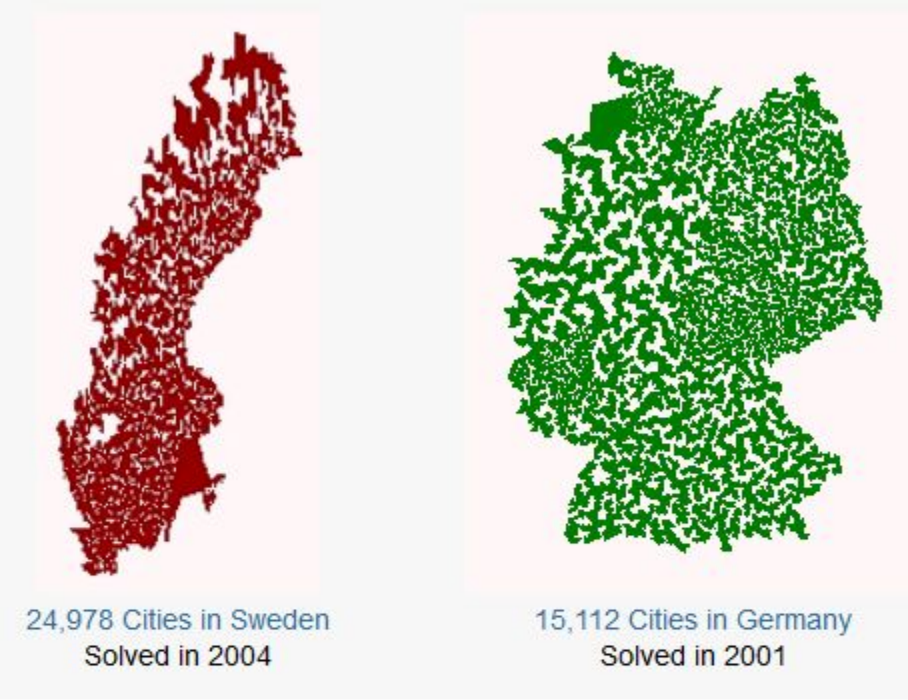


Motivating Example

Traveling salesman problem (TSP)

- Minimize distance traveled between all
- Each city visited once
- Final city is start city

May be symmetric (*bi-directional paths*) or as



NP-complete problem (NP and NP-hard)

- Easy to verify solution
- No easy way to figure out the ‘correct’ solution

Great for testing out optimization algorithms

Types of Search Algorithms

Hill climber

Simulated annealing

Particle swarm optimization

Ant colony optimization

Evolutionary computation

Full disclosure

There are many variants to each of these algorithms

- People tend to ‘like’ using algorithms they’re comfortable with
- May extend them to different domains

E.g., I like genetic algorithms so I’m going to use them everywhere

- Even when it doesn’t make sense
- Favorite tool is a hammer --- so I’m going to use it to fasten screws…

or a bolt…

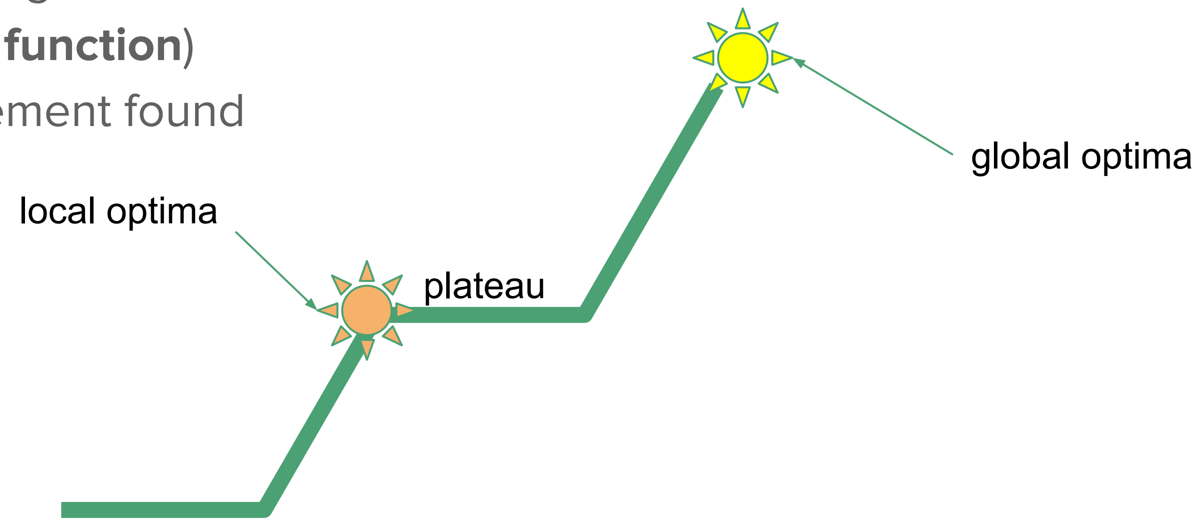
Hill Climber

“Simplest” optimization algorithm -- widely used as an initial attempt when solving a problem

1) Start with (random) solution to problem
2) Incrementally change *single* element
3) Check result
4) Repeat until no improvement found

Hill Climber

1) Start with (random) solution to problem
2) Incrementally change *single* element
3) Check result (**objective function**)
4) Repeat until no improvement found



Basically a greedy algorithm

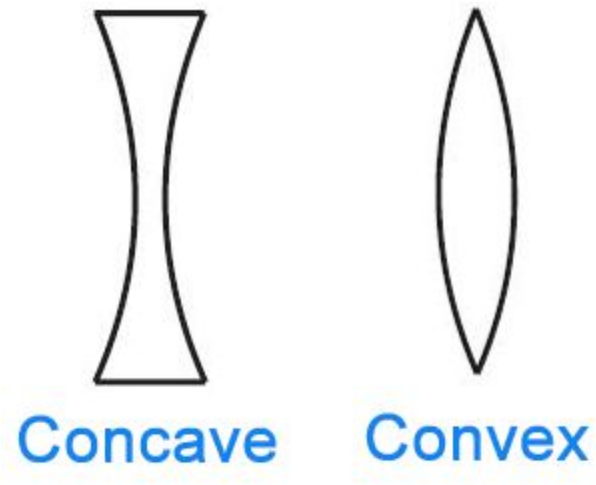
Hill Climber

Optimal for:

- Simple problems
- Problems where you don't care about "best"
  - Only valid!
- Convex problem space (why???)

Suboptimal for:

- Problems with multiple local optima



Hill Climber

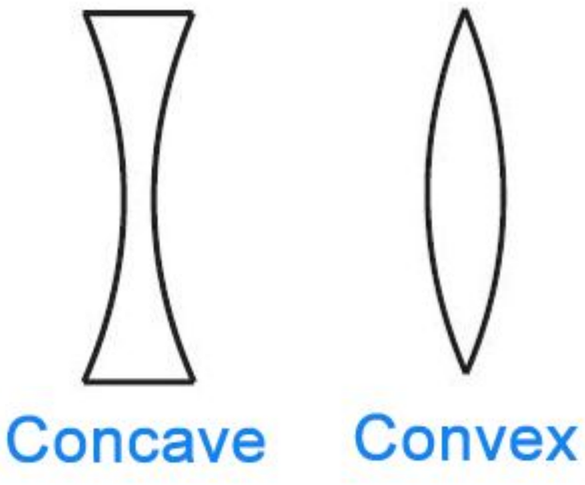
Optimal for:

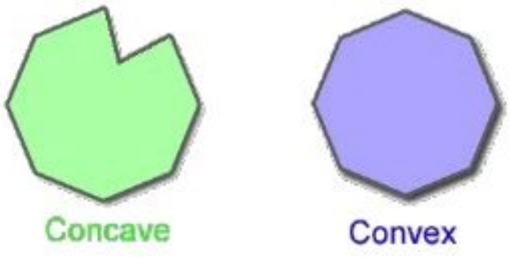
- Simple problems
- Problems where you don't care about "best"
    - Only valid!
- Convex problem space (why???)
    - **ONLY FINDS LOCAL OPTIMA**

Suboptimal for:

- Problems with multiple local optima

https://www.youtube.com/watch?v=kOFBnKDGtJM





Optimizing hill climbing

Random-restart hill climbing

- Restart with a new random solution
- Hopefully you’re in a better place

Hill Climbing and TSP

Solution = path between cities (this will be reflected in pretty much all algorithms)

Initialization

- Return tour of correct length with randomly ordered cities

Objective function

- Minimize the traveled distance WHILE ensuring that constraints are not violated
    - No city visited twice, start and end in same city, etc.

Hill Climbing and TSP

Example (% SO: https://stackoverflow.com/questions/14971181/hill-climbing-search-algorithm-applied-to-travelling-salesman)

Cities A-G

- Randomly assign first solution as ABCDEFGA
  - Pick neighbors for **first** iteration
    - A**CB**DEFGA, A**DCB**EFGA, A**ECDB**FGA, A**FCDEB**GA, A**GCDEFB**A
  - Compare fitnesses and keep the **best only**
  - Pick neighbors for **second** iteration
    - ...
  - Go until you haven't made any notable progress OR you're out of iterations

What kinds of problems in OOSAD could you optimize?

Tabu Search

Simulated Annealing

Another optimization algorithm

- Focus on avoidance of local optima
- Draws from metal annealing
    - Heating above recrystallization temperature
    - Cooling to acceptable level
    - Improves ductility (deform under stress) / reduce hardness

Uses a cost function

- How well the given solution performs

Factors in temperature (heating then cooling *solutions*)

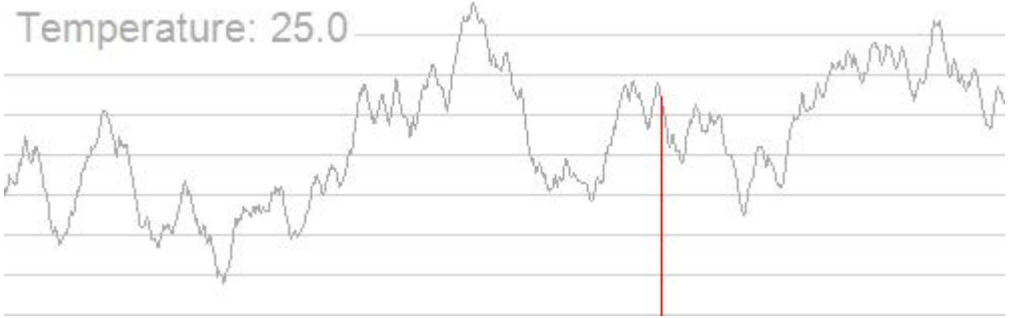
http://katrinaeg.com/simulated-annealing.html

Simulated Annealing

1) Generate random solution - c_{old}
   a) Can seed if you want
2) Calculate cost
   a) How “well” it performed
   b) Your fitness function
3) Generate random neighbor - c_{new}
   a) Should only have **one** parameter changed to make it a ‘neighbor’
   b) Calculate how well neighbor performed
4) Compare solutions (this is for minimization ➔ see next slide)
   a) if ($c_{old} > c_{new}$): move to new solution
   b) else: move to new solution (maybe?)

5) Repeat 2-4 until acceptable solution found / # of generations reached

Simulated Annealing



4) Compare cost

- Depends on your metric
- If you’re minimizing something, cost should be smaller
  - E.g., traveling salesman problem ➔ reduce number of steps
- If you’re maximizing something, cost should be larger
  - E.g., aggregate requirements satisfaction

Minimizing:

a) if ($c_{old} > c_{new}$): move to new solution
b) else: move to new solution (maybe?)

Why maybe? Why keep the “worse” solution?

Simulated Annealing

How does heat factor in?

Temperature ➜ relates to iteration you are at (similar to a GA's generation)

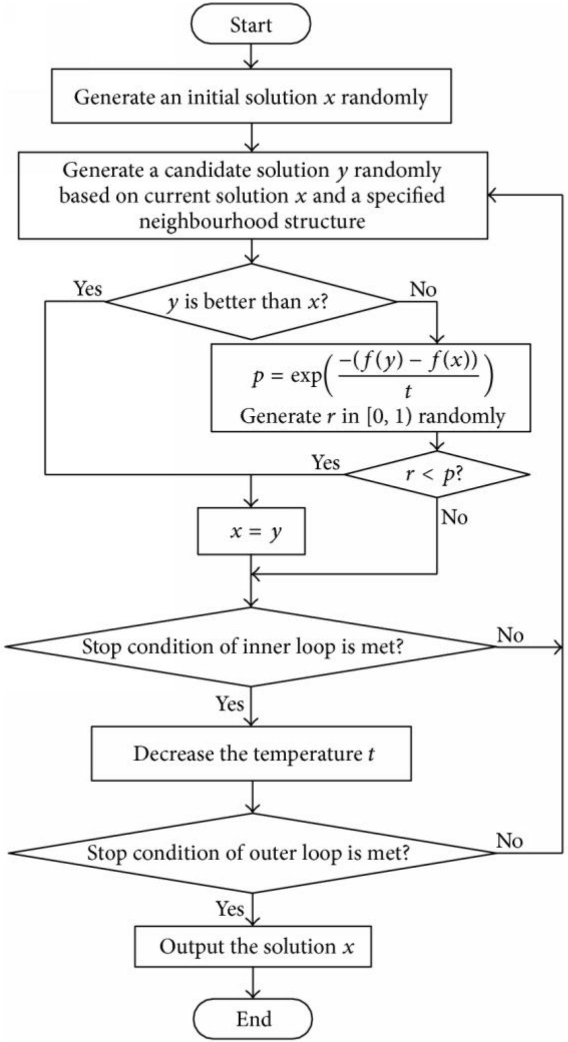
Intent is to go above the current solution and cool down to it

A random solution is to be 'annealed'

For each iteration, lower temperature slightly (depends on your value, decrement or decrease by a small decimal point)

Accepting a worse solution is an **acceptance probability**

```
1.0               if e'<e   # e' and e are the calculated cost
exp(-(e'-e)/T)    else      # (also called energy)
```



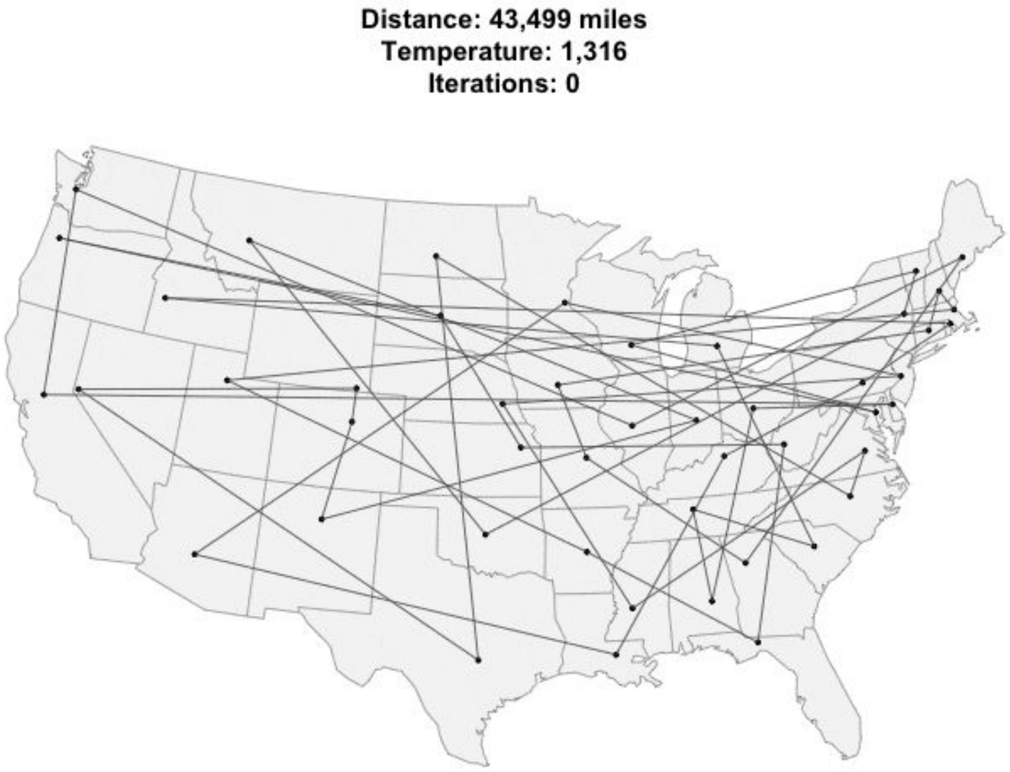
https://www.researchgate.net/publication/298209081/figure/fig7/AS:341632911200275@1458463041655/Flowchart-of-simulated-annealing-algorithm.png

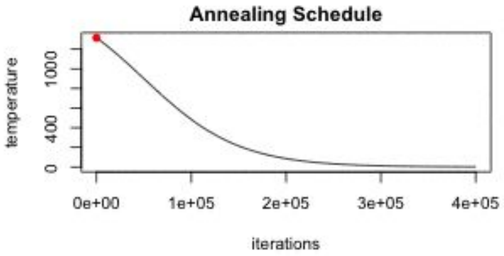
Simulated Annealing

Generally

- Temperature starts at 1.0
- Decreased at end of each iteration by multiplying by **α**
    - **α** is typically between 0.8 to 0.99

SA and TSP


Distance: 43,499 miles
Temperature: 1,316
Iterations: 0


Annealing Schedule
temperature
0
400
1000
0e+00
1e+05
2e+05
3e+05
4e+05
iterations

SA and TSP

1) Pick random initial tour
2) Pick random neighbor of existing tour
    a) Choose two cities at random, and reverse tour between (possibility)
3) Compare tours based on cost function
    a) Better? accept
    b) Worse?
        i) Calculate probability of accepting inferior tour
        ii) Factors in length and temperature of annealing process
            (1) Higher temperature → more likely to accept worse tour
4) Go back to (2) and repeat, lowering temperature

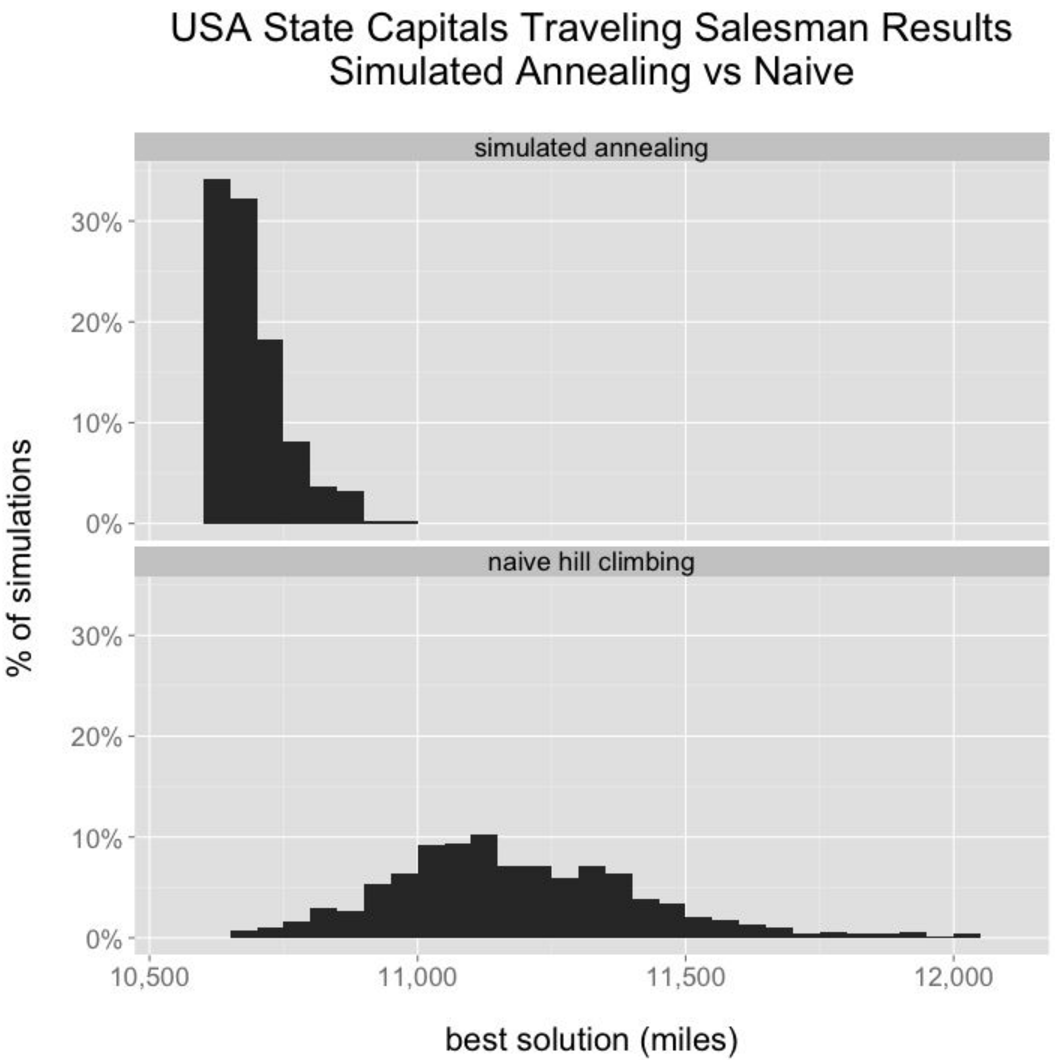
SA vs. HC

SA not a guarantee!

But does pretty good...

Web demo:

http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/



Particle Swarm Optimization (PSO)

Another “inspired by” algorithm

- Flocking patterns of birds // Schooling patterns of fish

And

- Another iterative technique

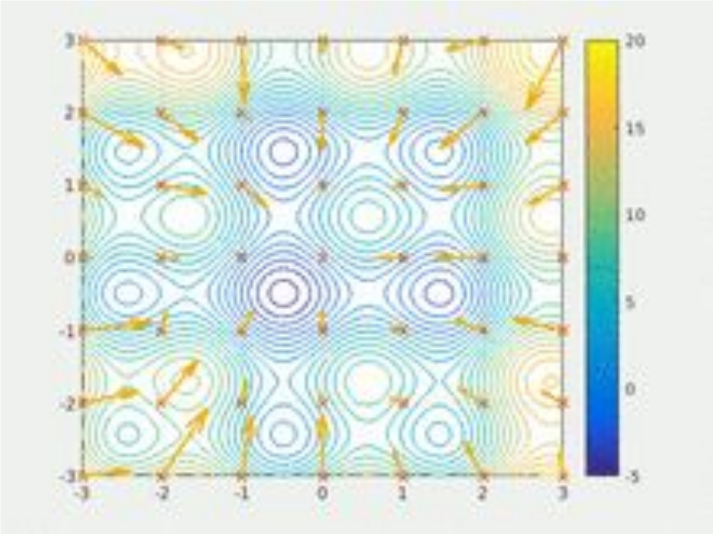
Group of solutions adjust closer to member whose objective function is “the best”

- Tightening pattern

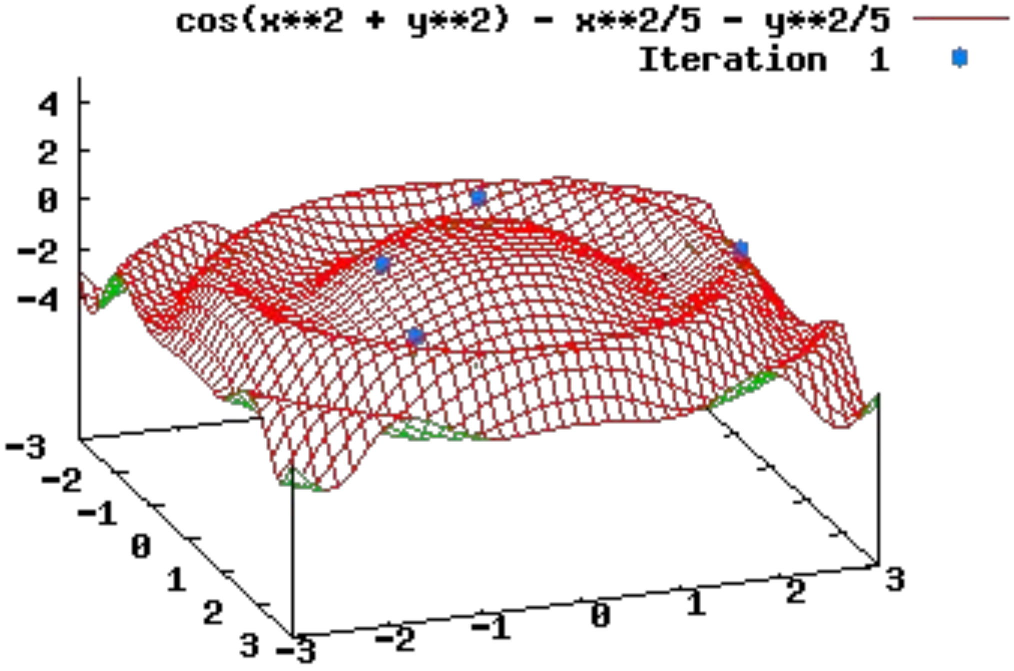
http://mnemstudio.org/particle-swarm-introduction.htm / http://www.swarmintelligence.org/tutorials.php

Birdfriends

PSO



PSO



PSO

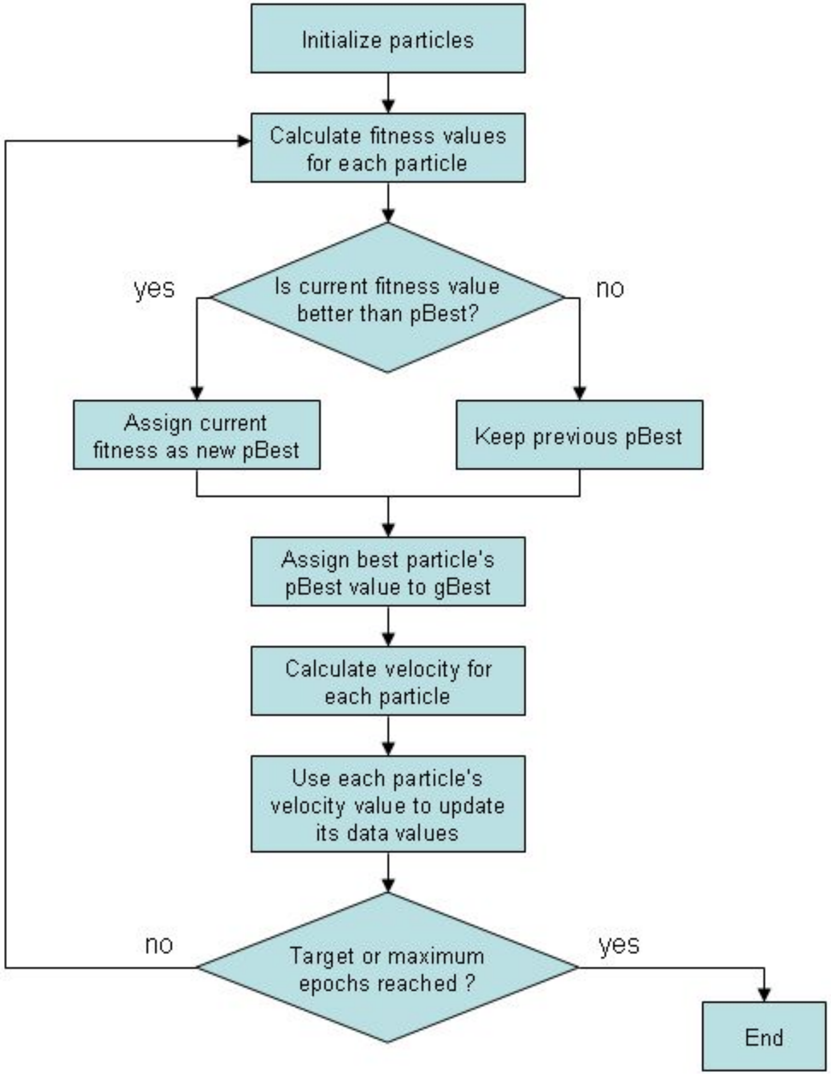
Each particle comprises:

- Solution representation (encoding)

- Velocity (distance from target) // Target ➜ **current optima**
    - How fast a bird needs to catch up to the target
    - The further away, the larger the velocity
    - Bird example ➜ how far away we are from food
    - Pattern recognition ➜ how different you are from the actual value

- Record of ‘personal best’ ➜ pBest

PSO

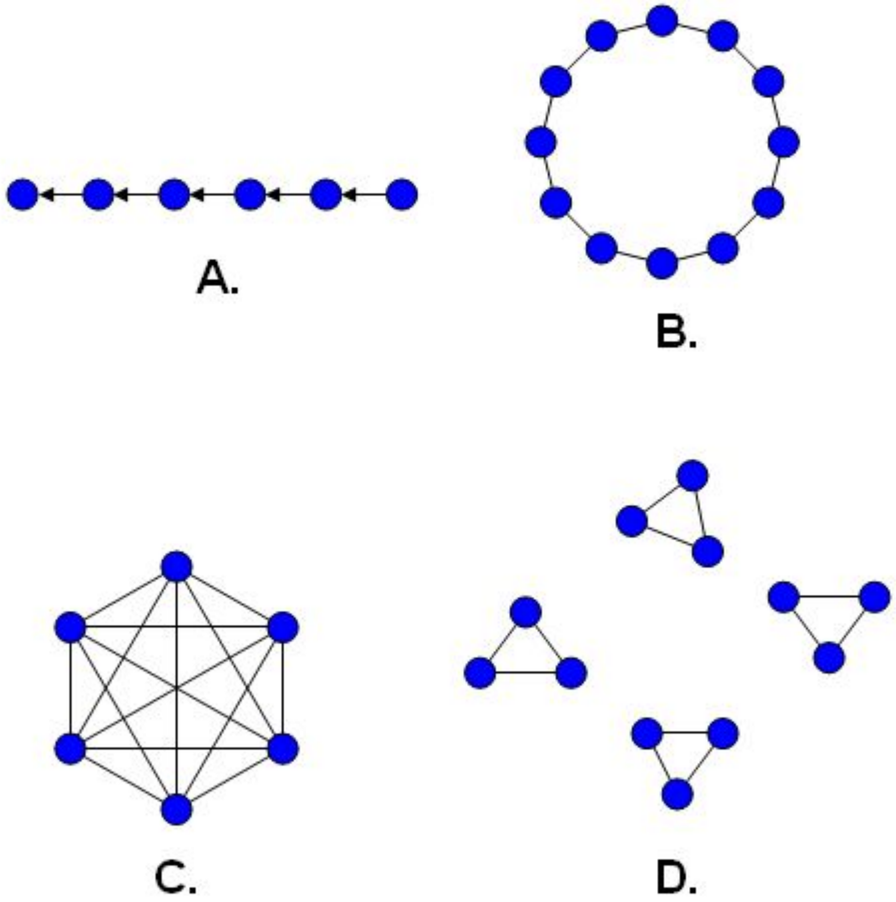

Initialize particles
Calculate fitness values for each particle
Is current fitness value better than pBest?
yes
no
Assign current fitness as new pBest
Keep previous pBest
Assign best particle's pBest value to gBest
Calculate velocity for each particle
Use each particle's velocity value to update its data values
Target or maximum epochs reached ?
no
yes
End

PSO

Particles may live in neighborhoods

- Avoid local optima

Common topologies ➔



A. Only compare self to next-best
B. Compare only to left/right
C. Compare to all
D. Compare to local

PSO

How many particles?

- YMMV
- 10-20 is usually acceptable (http://www.swarmintelligence.org/tutorials.php) however may be domain-dependent

Addendum:

- Velocity may have a max cap
    - i.e., particles can't move faster than X
    - If velocity between source and target is too high, velocity will be capped
        - Birds can't move faster than speed of sound
            - Unless if it is an unladen swallow

PSO and TSP

Define equations for velocity and position of particles

- velocity (V) incorporates inertia and influence of other particles
- position (X) incorporates velocity and current position

1) Create population of particles with positions drawn from random distribution
   a) Can be uniform, Gaussian, etc.
2) Update velocities according to V
3) Move particles according to X
4) If position is better than prior, update

Here, position is the goodness of the solution (minimizing length, etc.)

No good visuals, sorry!

Ant Colony Optimization (ACO)

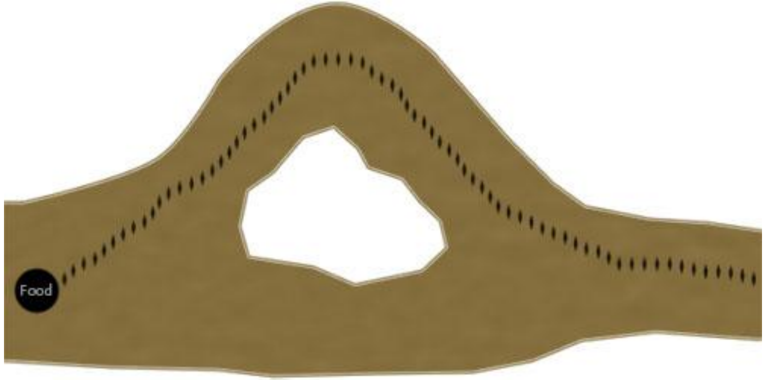
Inspired by...

ACO

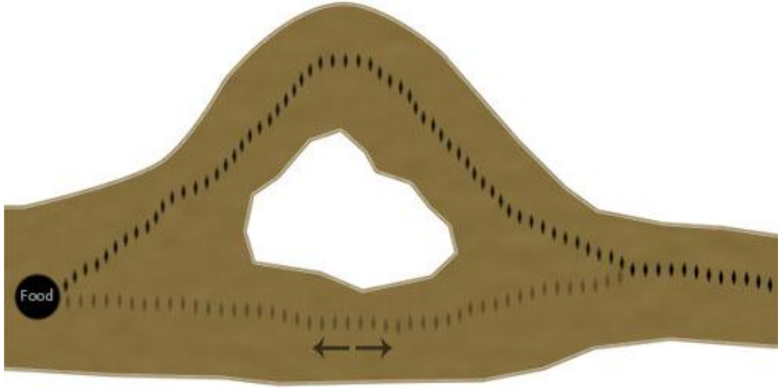
Why ants?

- Searching for food
- Basic rules
  - Pheromone trails → Stigmergy (act of laying down trail)
    - Communicate to other ants that food has been discovered
    - Ants find trails and decide to follow (or not)
    - Strength of pheromone a consideration
    - Evaporate over time (generally a few minutes)

ACO Pheromone Trail

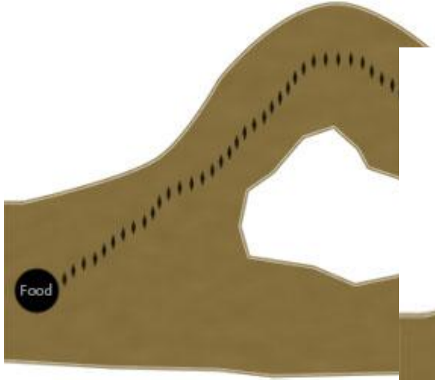


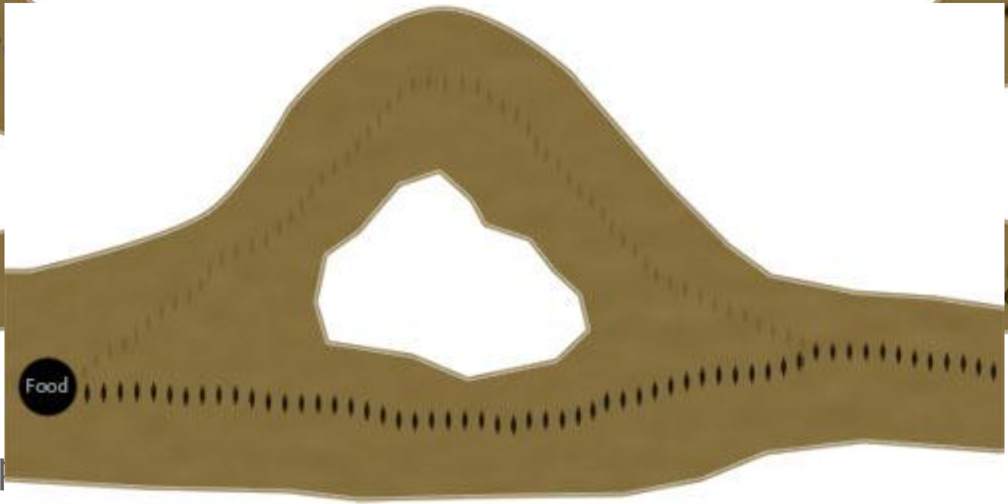
vs.

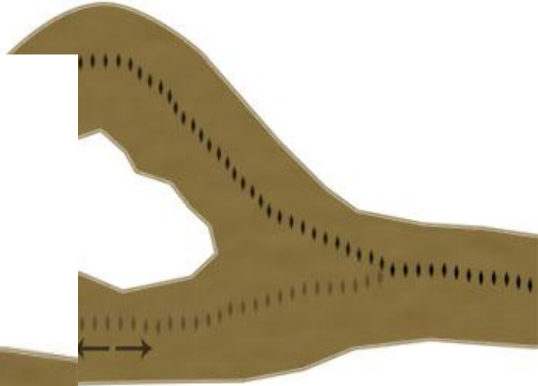


- Left image ➔ path discovered first, so ants start following it
- Right image ➔ shorter route discovered, ants start following that
  - Less strong than other trail, but that will fluctuate over time

ACO Pheromone Trail







- Left image → p
- Right image → shorter route discovered, ants start following that
    - Less strong than other trail, but that will **fluctuate** over time

ACO

Optimization considerations:

- Limited memory available
- Sense environment around “ant”
  - Not simply by pheromone
- May have a local search algorithm
  - Hybridized algorithm

http://staff.washington.edu/paymana/swarm/stutzle99-eaecs.pdf

ACO and TSP

1) *m* ants generated and each are placed at random cities
2) Each ant constructs a valid tour
    a) At city *i*, ant chooses unvisited city *j* probabilistically (based on pheromone strength) and length of path
        i) Prefer cities that are closer with high pheromone strength
        ii) Each ant has limited memory (tabu list)
            (1) Partial tour stored → guarantees valid solution
3) Pheromones updated after all tours constructed
    a) Lower strengths by constant
    b) Ants spread pheromones on paths taken
4) Repeat until...?

https://www.youtube.com/watch?v=eVKAIufSrHs

Break

http://evolve-a-robot.github.io/