CS4710: Artificial Intelligence
Local Search

A different kind of search problem ☺

Classical Search Vs. Local Search

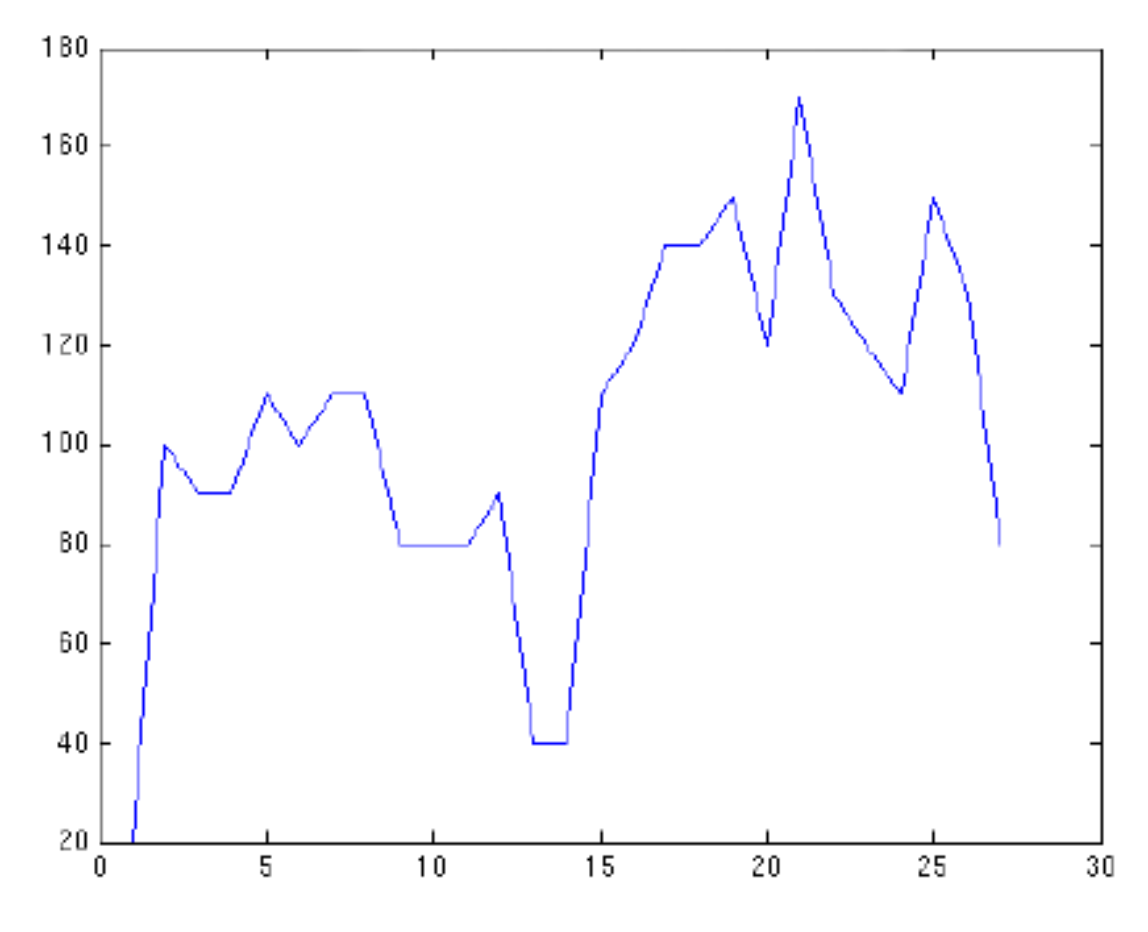
Classical Search

- Observable
- Deterministic
- Known Environment
- Solution of sequence of actions

Local Search

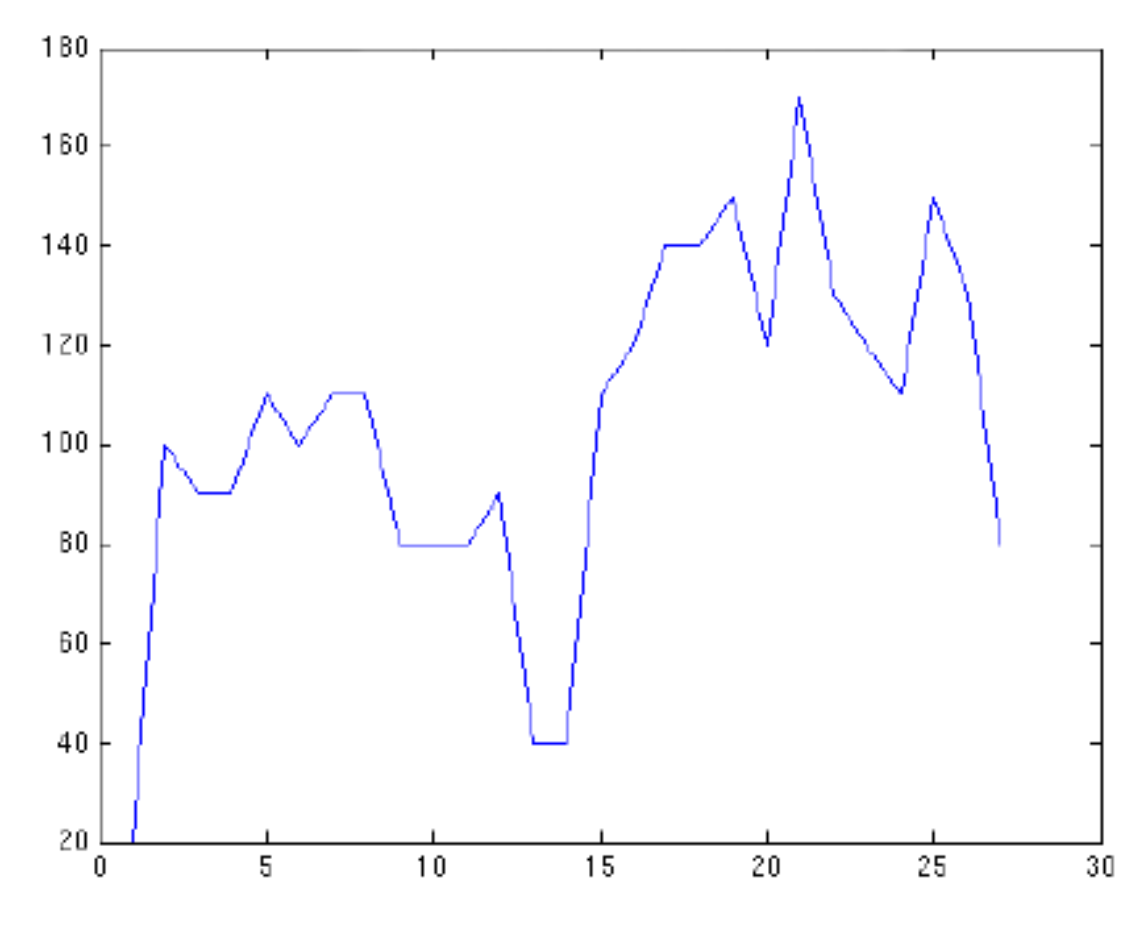
- All that matters is the solution state
- Don’t care about solution path
- Impossible to search whole space (too large)
- Often continuous search space

Local Search: Characteristics/Advantages



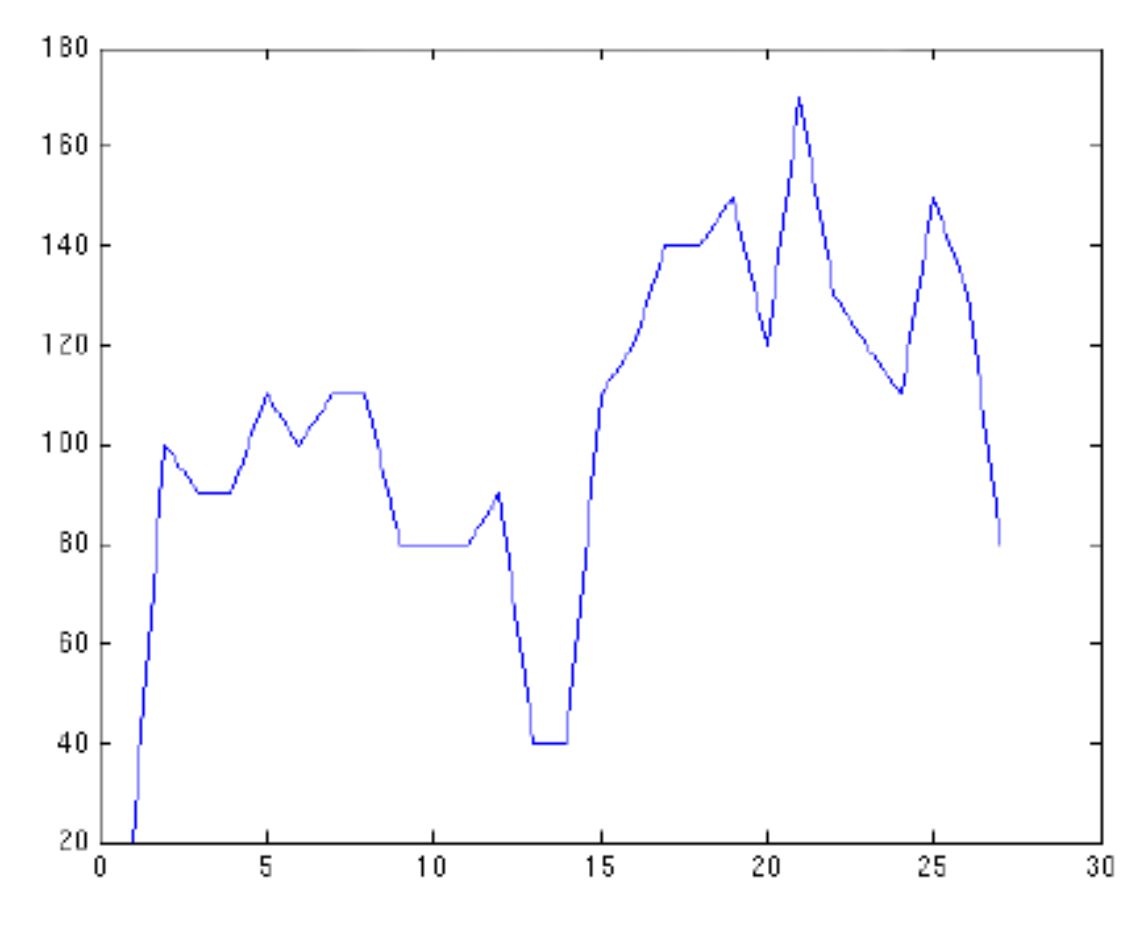
- Sometimes continuous search space (though not always)
- Concept of ‘neighboring’ solution states
- Very Little Memory
  - Usually Constant
- Can often find reasonable solutions in infinite (continuous) state spaces

Local Search: Characteristics/Advantages



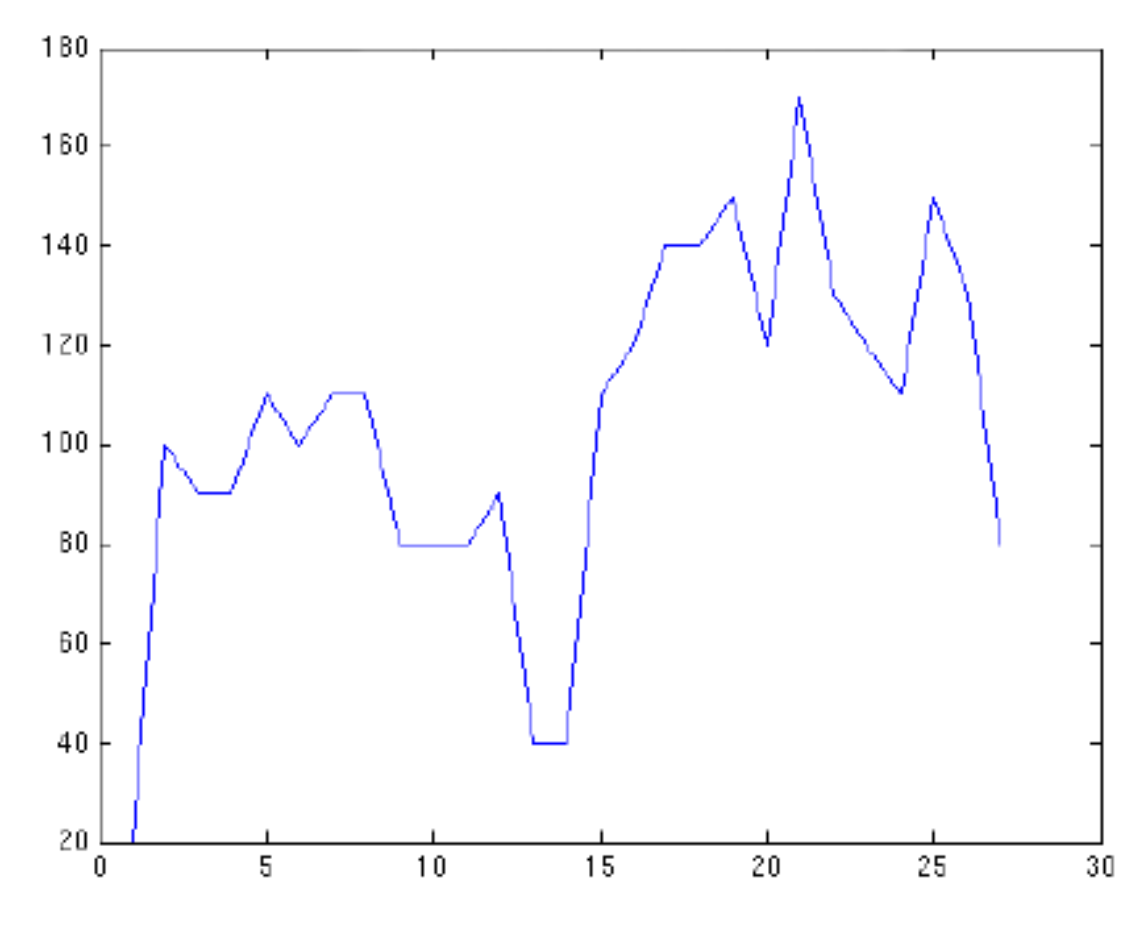
- Usually no ‘start state’. We just generate one however we want.
- Search space is usually VERY large, so not possible to search everything.

Local Search: Random Walk



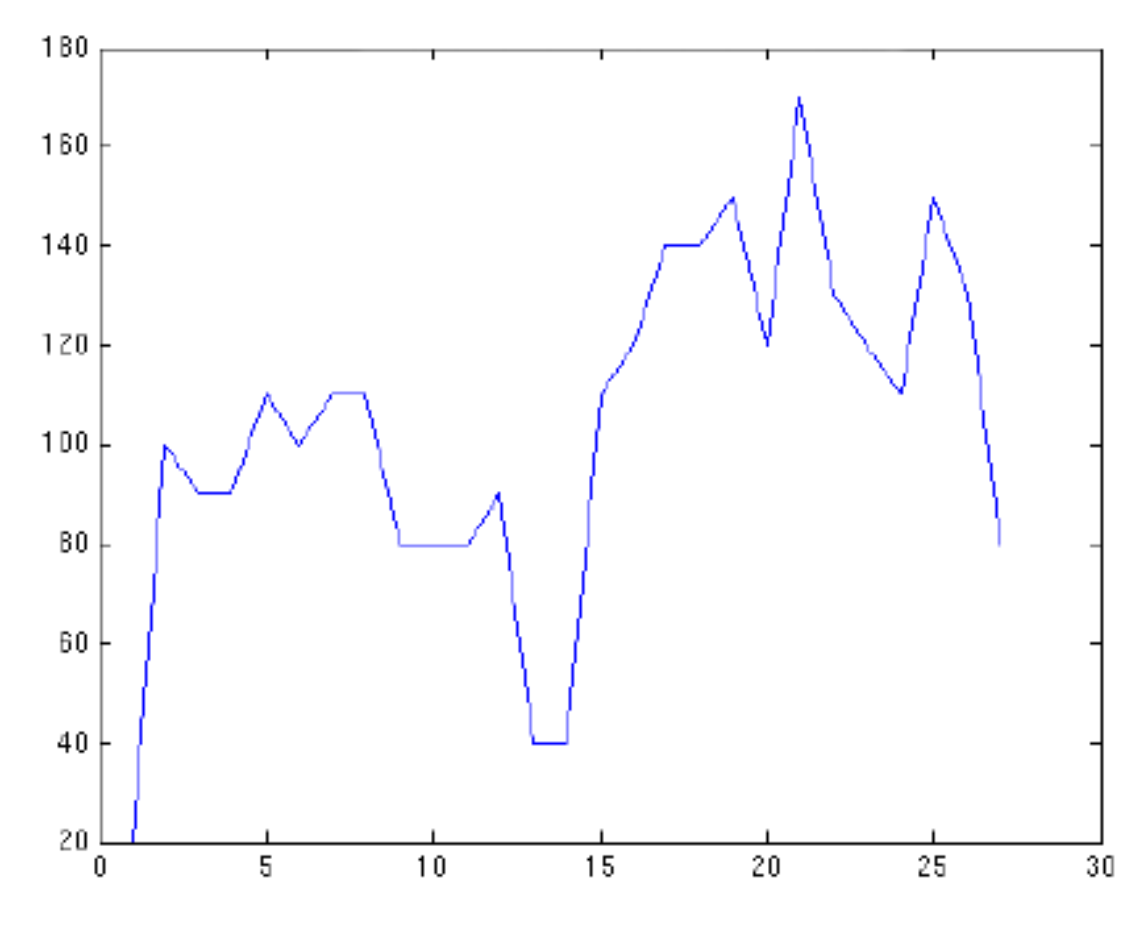
- 1. Pick a random point
- 2. Look at your neighbors
- 3. Move in a random direction
- Issues?

Local Search: Random Walk Improvement



- 1. Pick ‘x’ random points
  - Maybe 1000 or more?
- 2. Look at your neighbors of each
- 3. Each moves in a random direction
- Issues?

Local Search: Greedy Algorithm



- 1. Pick a random point
- 2. Look at your neighbors
- 3. Keep going up until you find the maximum
- Issues?

Random Walk vs. Hill Climbing

Random Walk

- Walks around randomly, so very good at exploring the search space.
- Very good at finding the various “hills” in the search space.
- NOT good at climbing up hills or optimizing any promising solution states.

Hill Climbing

- Good at optimizing a solution state, due to climbing nature.
- BAD at exploring various parts of search space.
- Often stuck in local maxima.

Random Walk vs. Hill Climbing

Pseudo code for both algorithms

Given current state S

Generate neighbor state S':

    if (util(S') > util(S)) move with probability p;

    else move with probability 1-p;

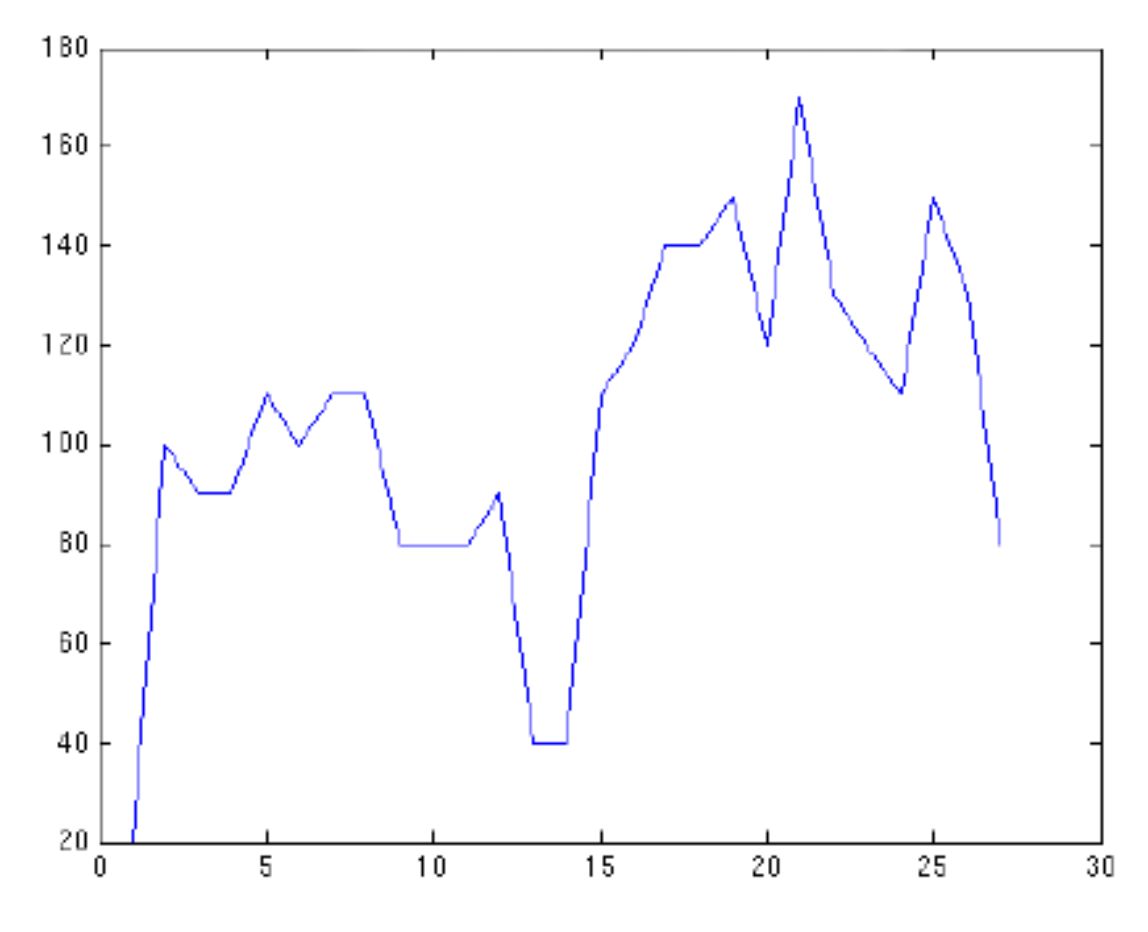
Repeat constant number of times

//For hill-climbing, p=1.0 For random-walk, p=0.5

//so 'p' is what gives us control over algorithm's behavior…hmmmmmm

Simulated Annealing

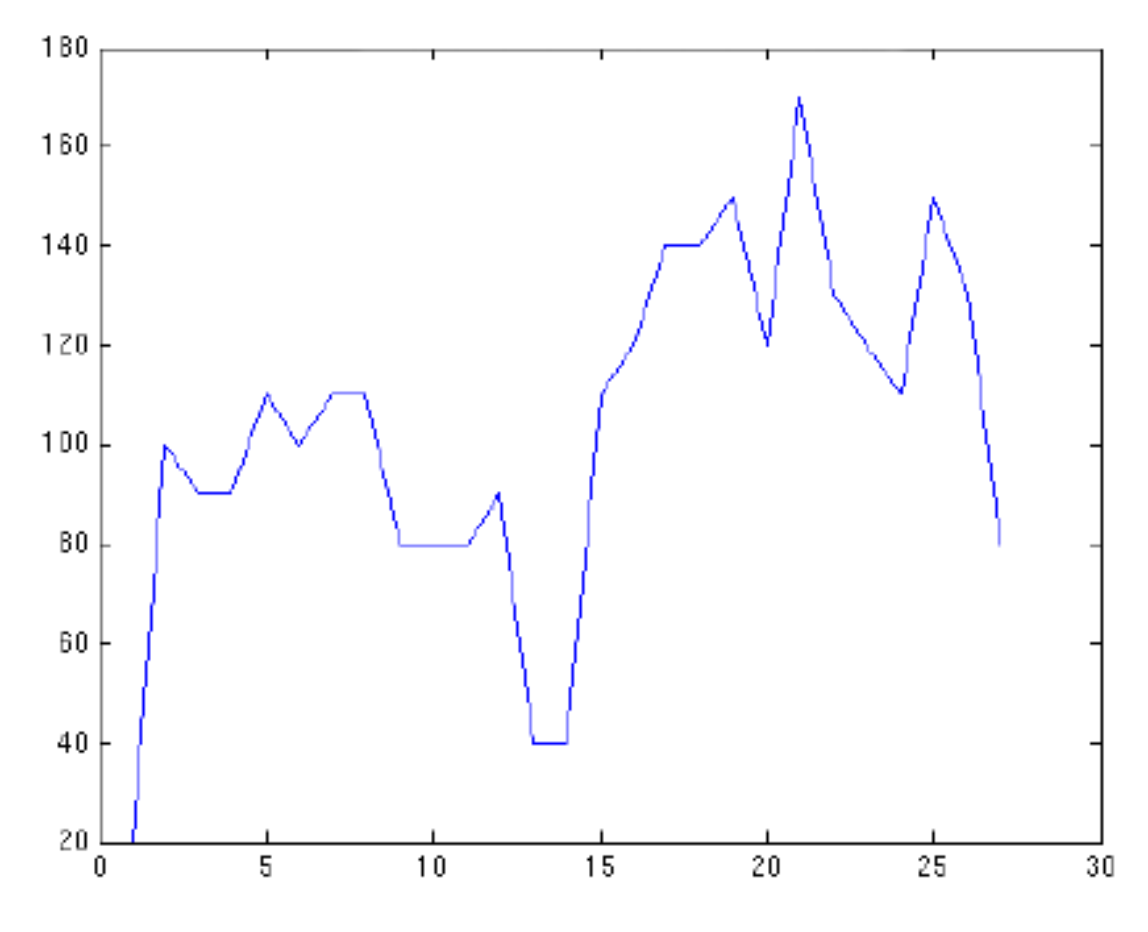
Annealing



- Annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low-energy crystalline state.

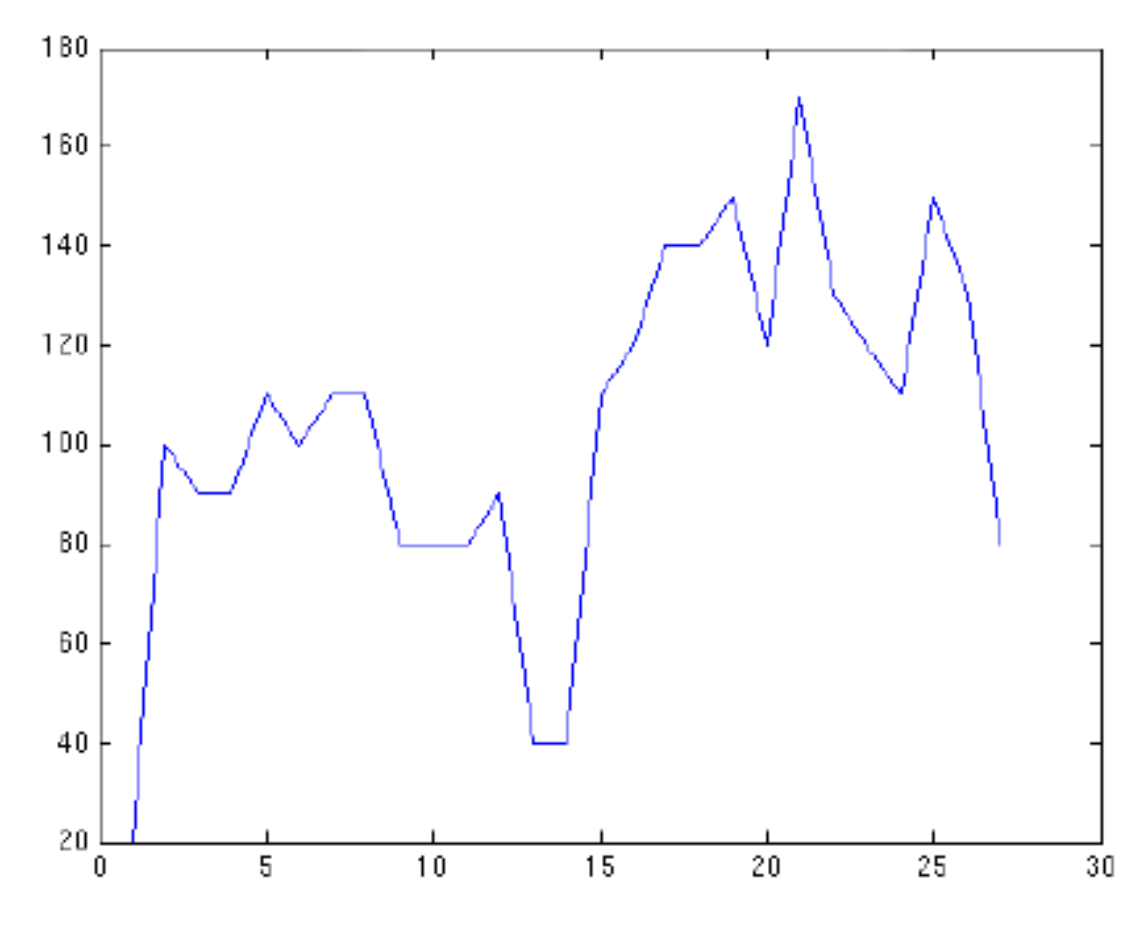
- Simulated annealing is just a search algorithm motivated by this idea.

Annealing



- Idea!
- What if system has a ‘temperature‘
- At high temps, system explores much more, but at low temps, is more conservative (closer to hill-climbing)

Simulated Annealing



1. Choose random initial state, high initial temperature, and cooling rate
2. Choose random neighbor state
3. Randomly decide whether to move to neighbor based on temperature
    1. Better neighbors have higher probability of being moved to. More detail later.
4. Reduce temperature
5. Repeat steps 2-5 until cooled

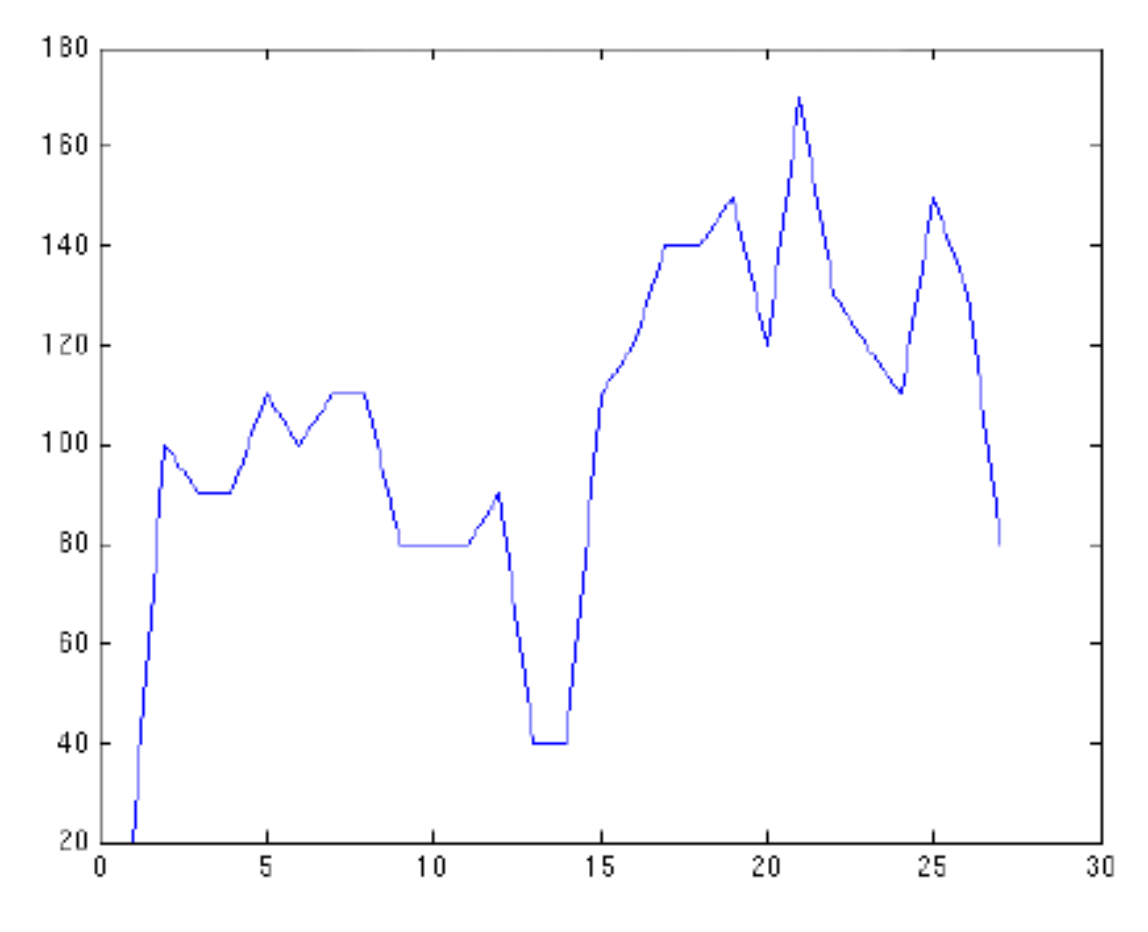
Simulated Annealing

```
// Loop until system has cooled
while (temp > 1) {
    Tour newSolution = randomNeighbor(currentSolution);

    // Get energy of solutions
    int currentEnergy = currentSolution.getDistance();
    int neighbourEnergy = newSolution.getDistance();

    // Decide if we should accept the neighbour
    if (acceptanceProbability(currentEnergy, neighbourEnergy, temp) > Math.random()) {
        currentSolution = new Tour(newSolution.getTour());
    }

    // Keep track of the best solution found
    if (currentSolution.getDistance() < best.getDistance()) {
        best = new Tour(currentSolution.getTour());
    }

    // Cool system
    temp *= 1-coolingRate;
}
```

Simulated Annealing

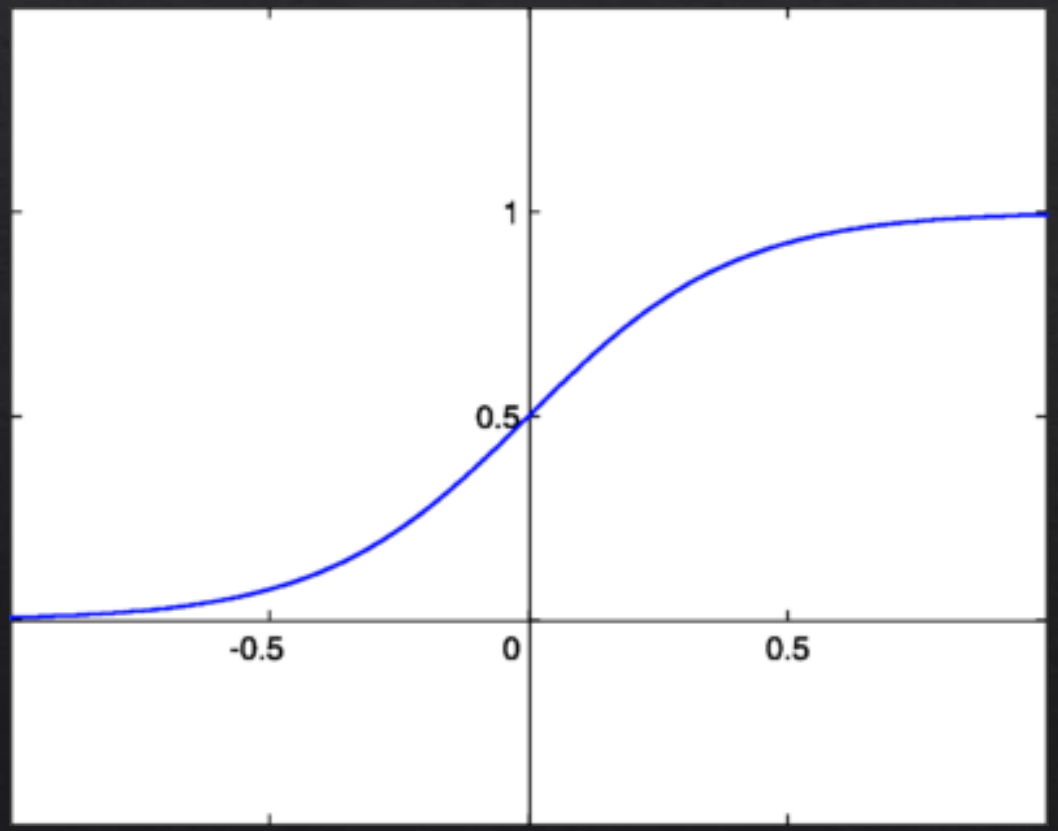
Example Animation (Wikipedia):
Simulated Annealing In Action

Simulated Annealing: Probability Function



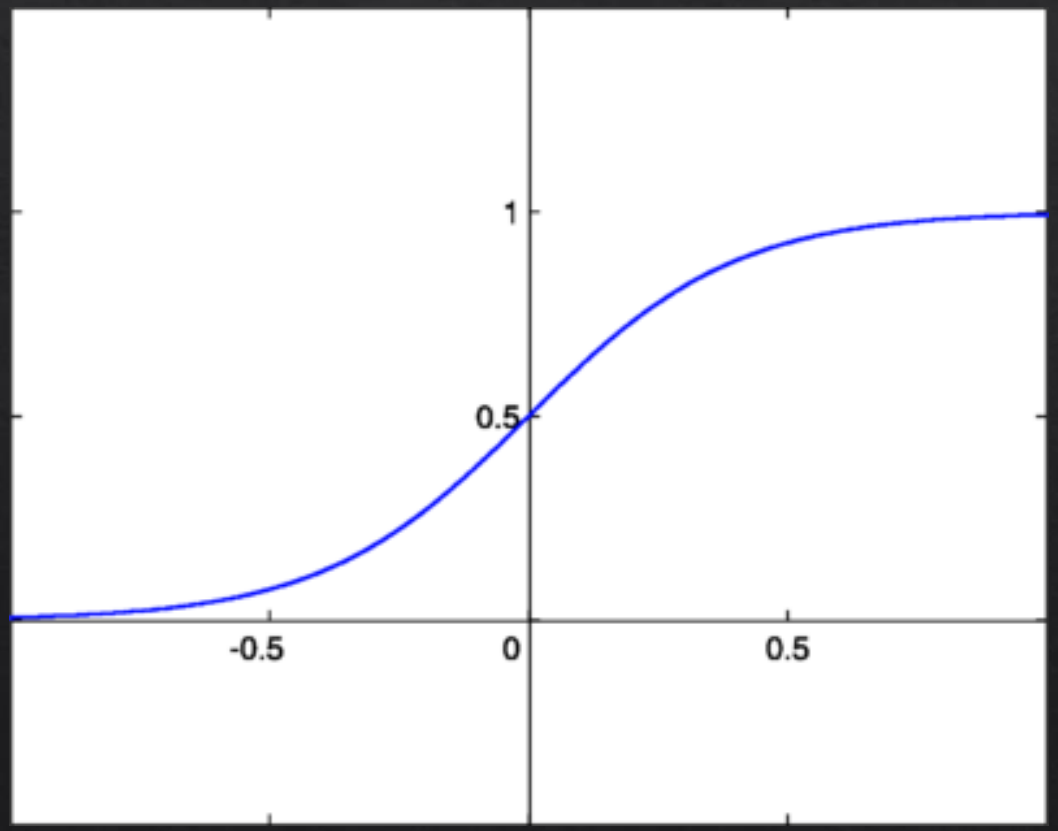
- Requirements:
  - Needs to be a function of ΔE where:
    - ΔE = Val(newState) – Val(curState)
    - Needs to give higher probability of moving to higher jumps
  - Needs to be a function of the temperature, which starts high and cools over time.
    - More likely to move to a worse state at higher temperatures.
    - As temp cools, becomes more conservative

Simulated Annealing: Sigmoid Function



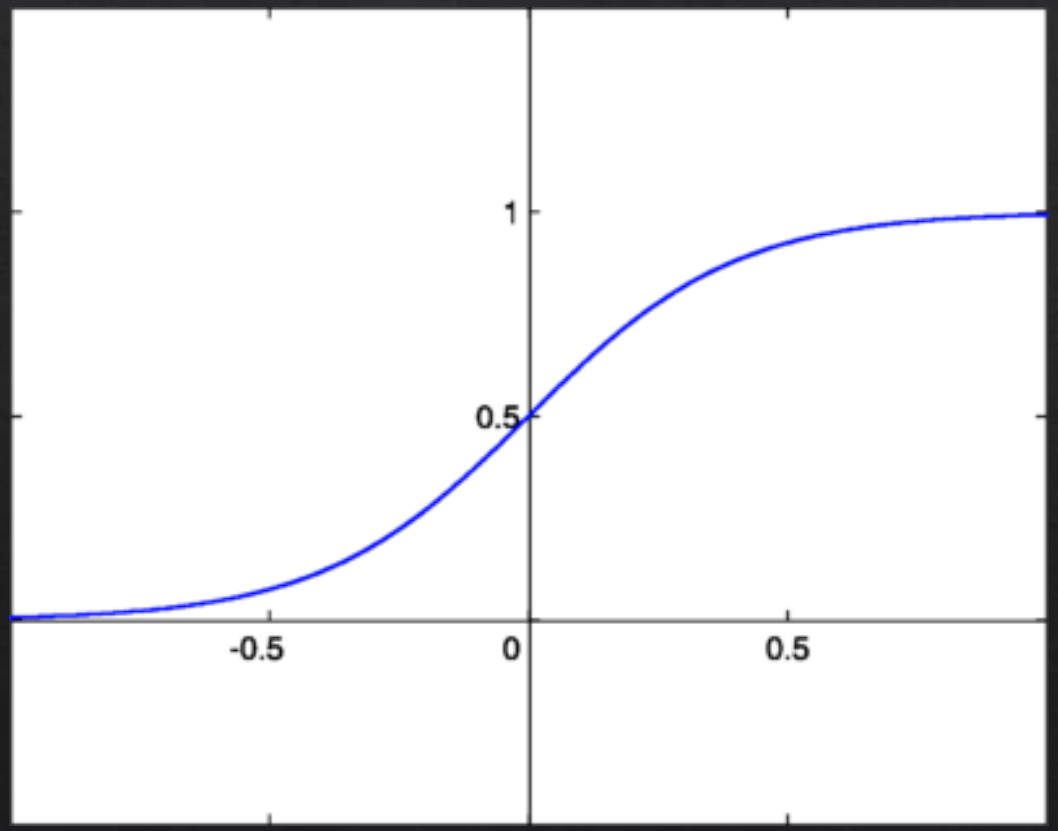
- $Y = 1 / (1 + e^{-X})$
- First attempt:
  - $X = \Delta E$

Simulated Annealing: Sigmoid Function



- $Y = 1 / (1 + e^{-X})$
- Second attempt:
  - $X = \Delta E / T$

Simulated Annealing: Sigmoid Function

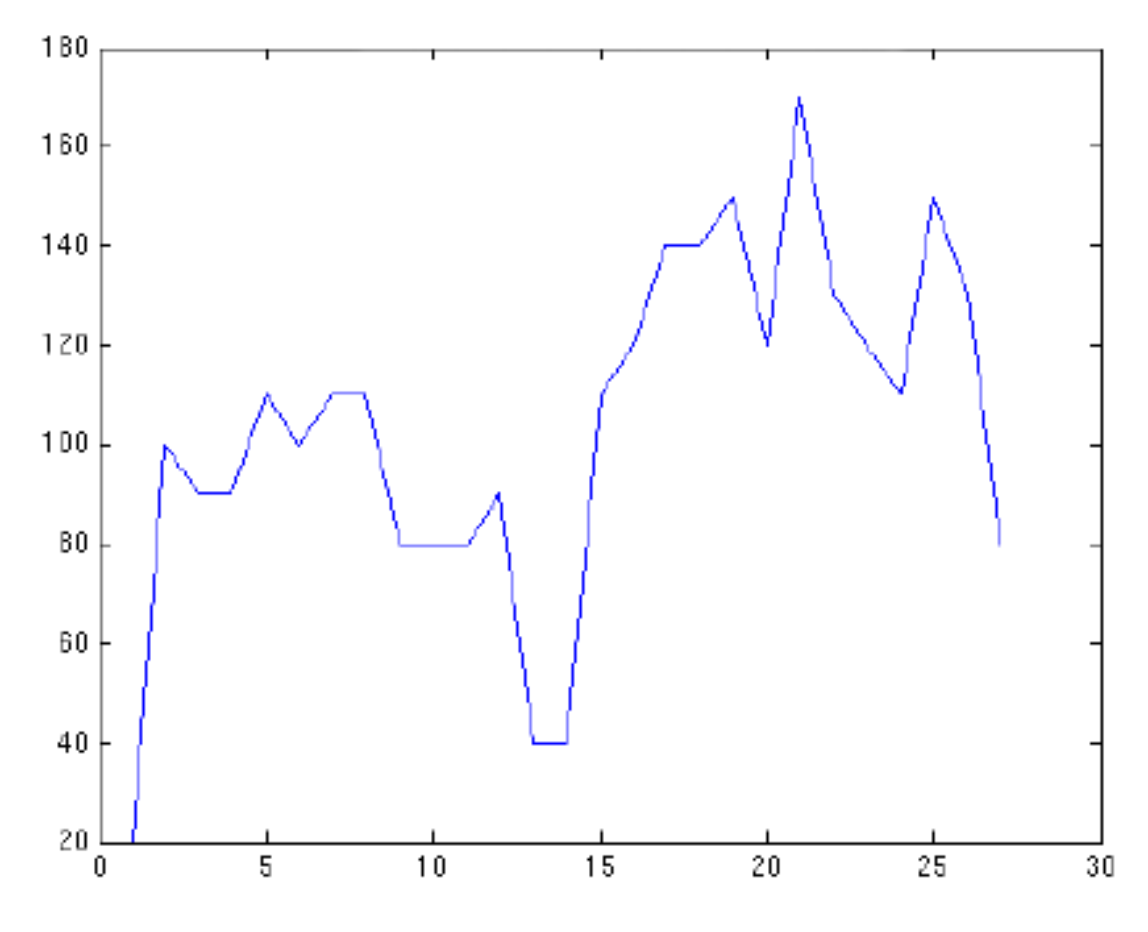


- $Y = 1 / (1 + e^{-X})$
- $X = \Delta E / T$
- How does ΔE affect P?
- How does T affect P?

Simulated Annealing

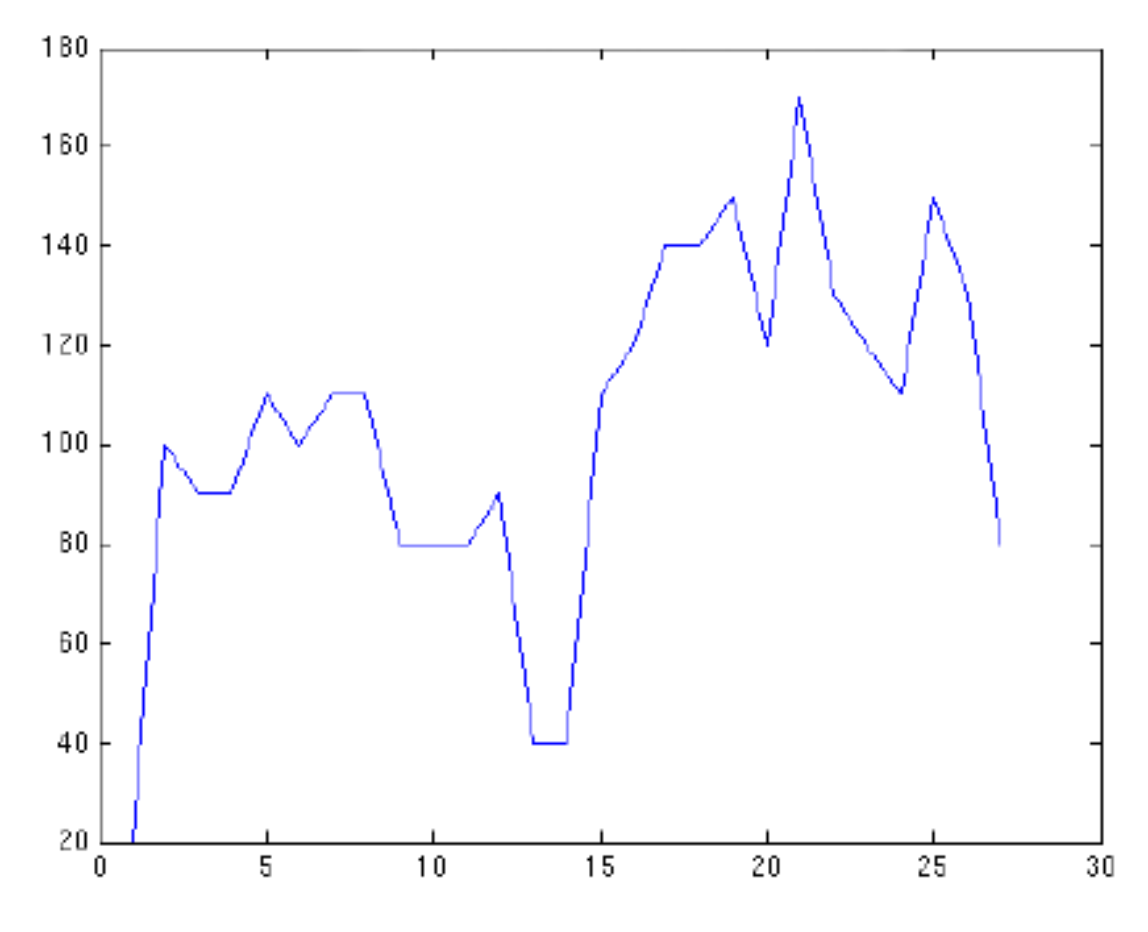
```
// Loop until system has cooled
while (temp > 1) {
    Tour newSolution = randomNeighbor(currentSolution);

    // Get energy of solutions
    int currentEnergy = currentSolution.getDistance();
    int neighbourEnergy = newSolution.getDistance();

    // Decide if we should accept the neighbour
    if (acceptanceProbability(currentEnergy, neighbourEnergy, temp) > Math.random()) {
        currentSolution = new Tour(newSolution.getTour());
    }

    // Keep track of the best solution found
    if (currentSolution.getDistance() < best.getDistance()) {
        best = new Tour(currentSolution.getTour());
    }

    // Cool system
    temp *= 1-coolingRate;
}
```

Choosing a Cooling Rate



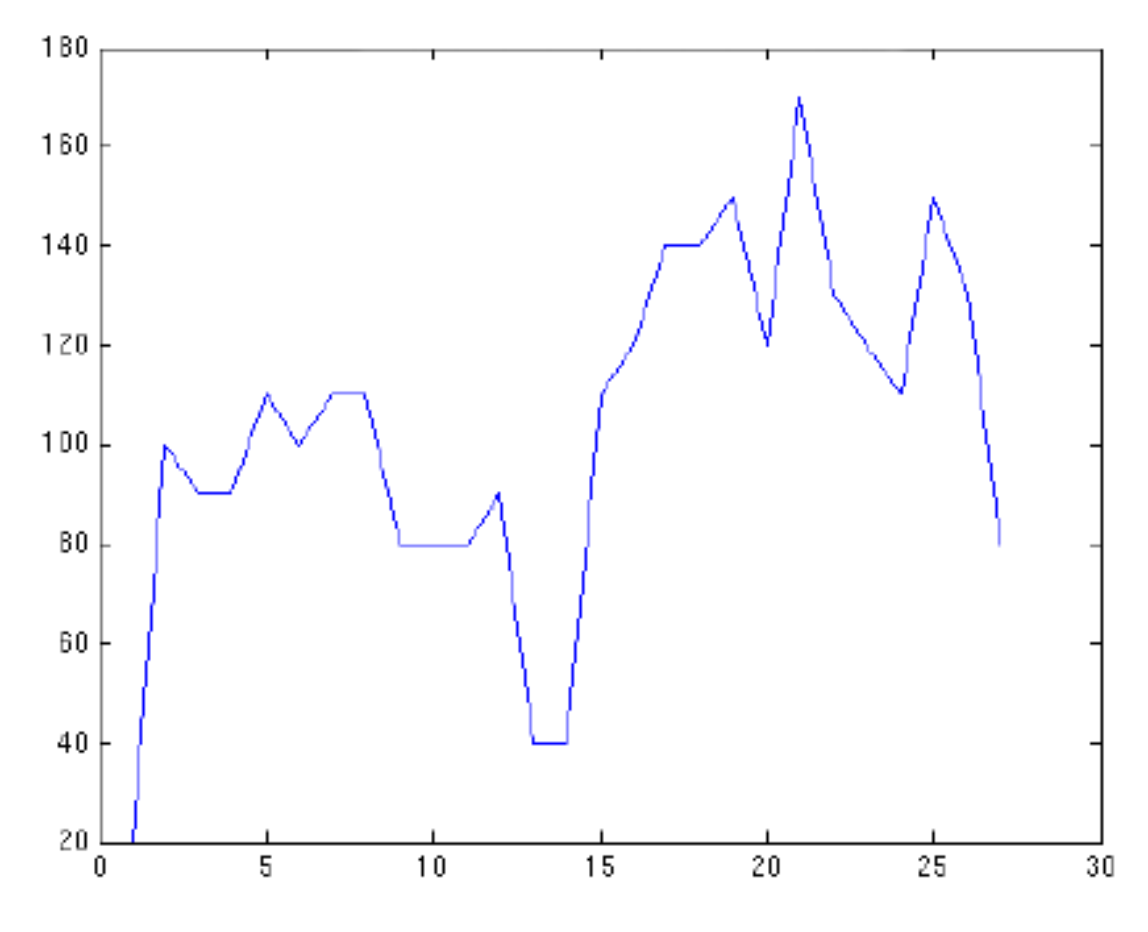
- No silver bullet
- Generally, high temperature and low cooling rate are best

Simulated Annealing: Advantages



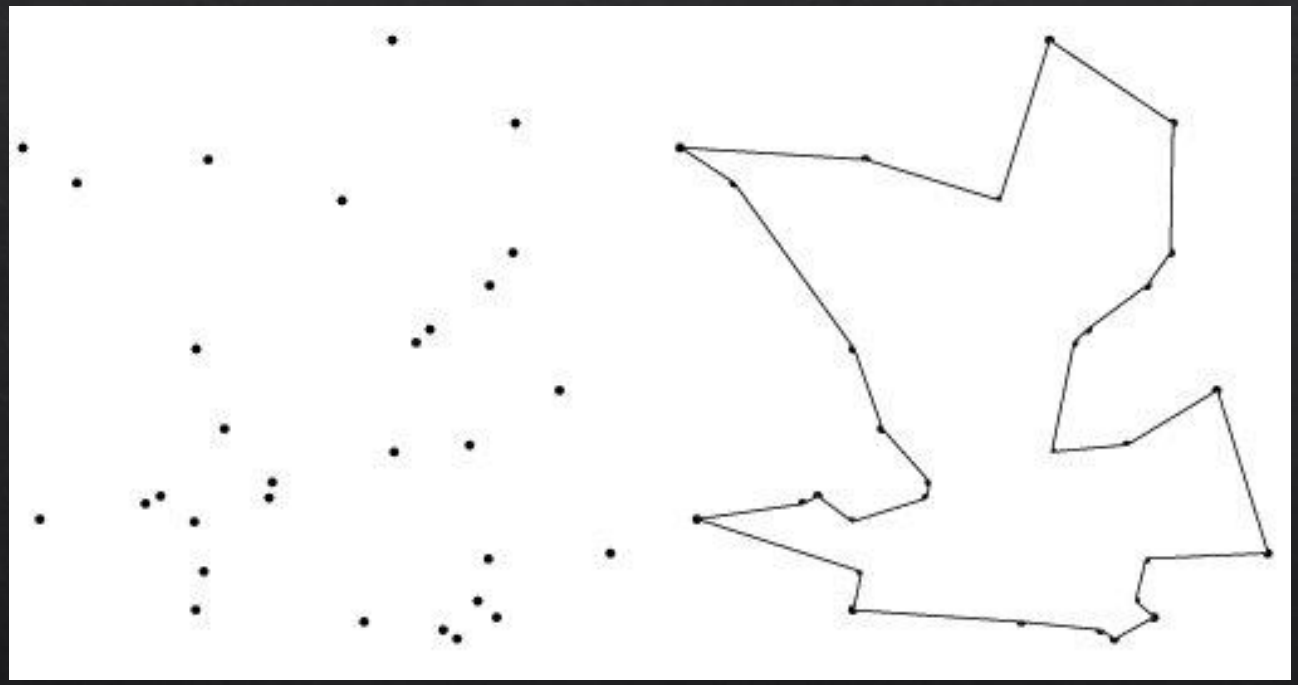
- Can deal with arbitrary systems and cost functions
- Is relatively easy to code, even for complex problems
- Generally gives good solutions
- Optimal? No! But often will return optimal solution in practice

Simulated Annealing: Complexity



- O(1) Time
  - Why?
- O(1) Space
  - Why?

Simulated Annealing: TSP

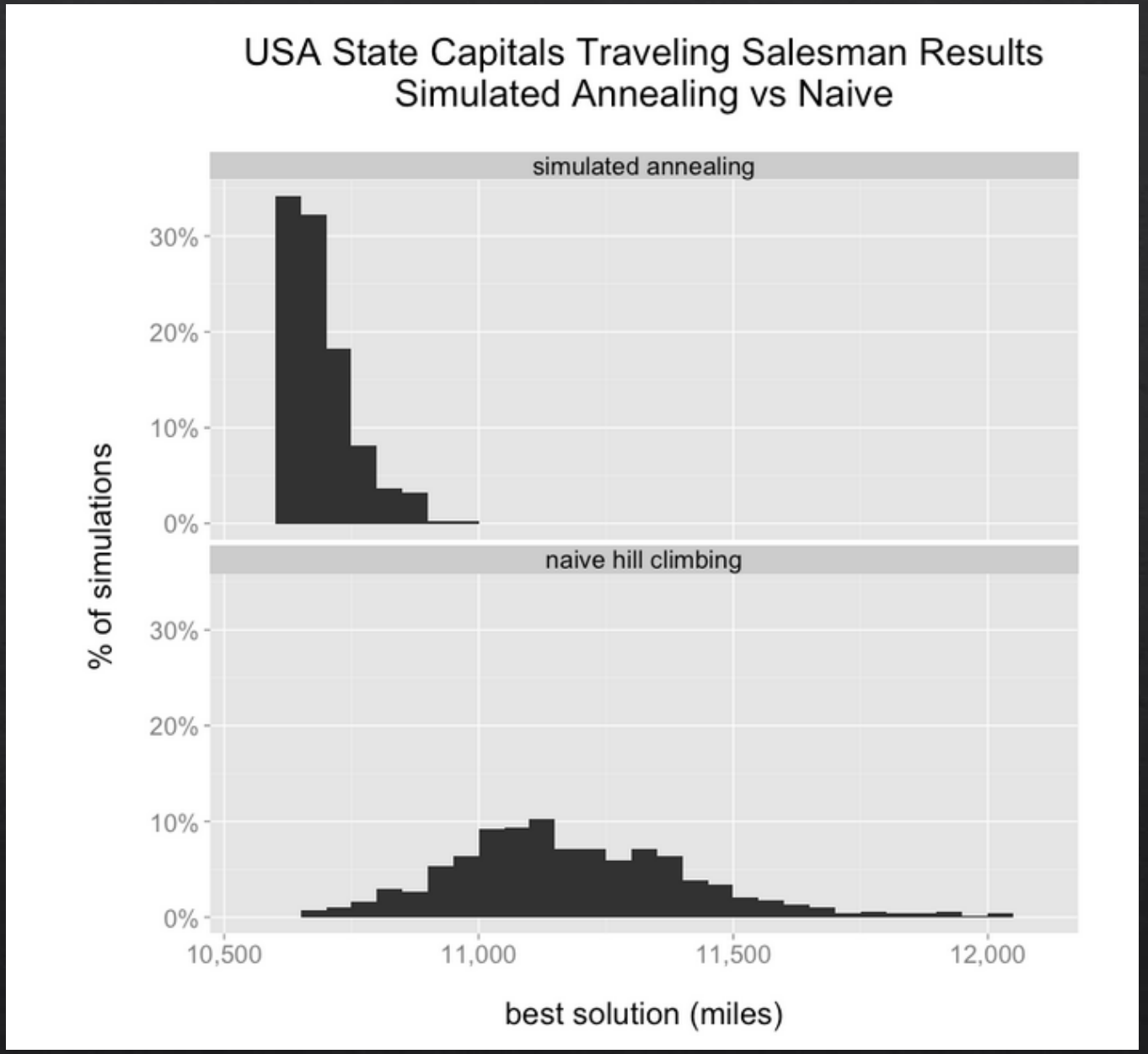


- We all know the Traveling Salesperson Problem.
- This problem is NP-Complete
- …but we can use SA to find very good solutions (not always optimal though).
- How would we do this?

Simulated Annealing: TSP

- Choose a random tour (i.e., a random order for the cities)
- Pick a new candidate tour by finding a random neighbor tour (how to find this?)
- Use sigmoid function to determine if you will follow that path or not
  - Function of temperature and difference in costs of tours
- Keep track of best tour seen so far
- Cool the temperature a bit
- Repeat

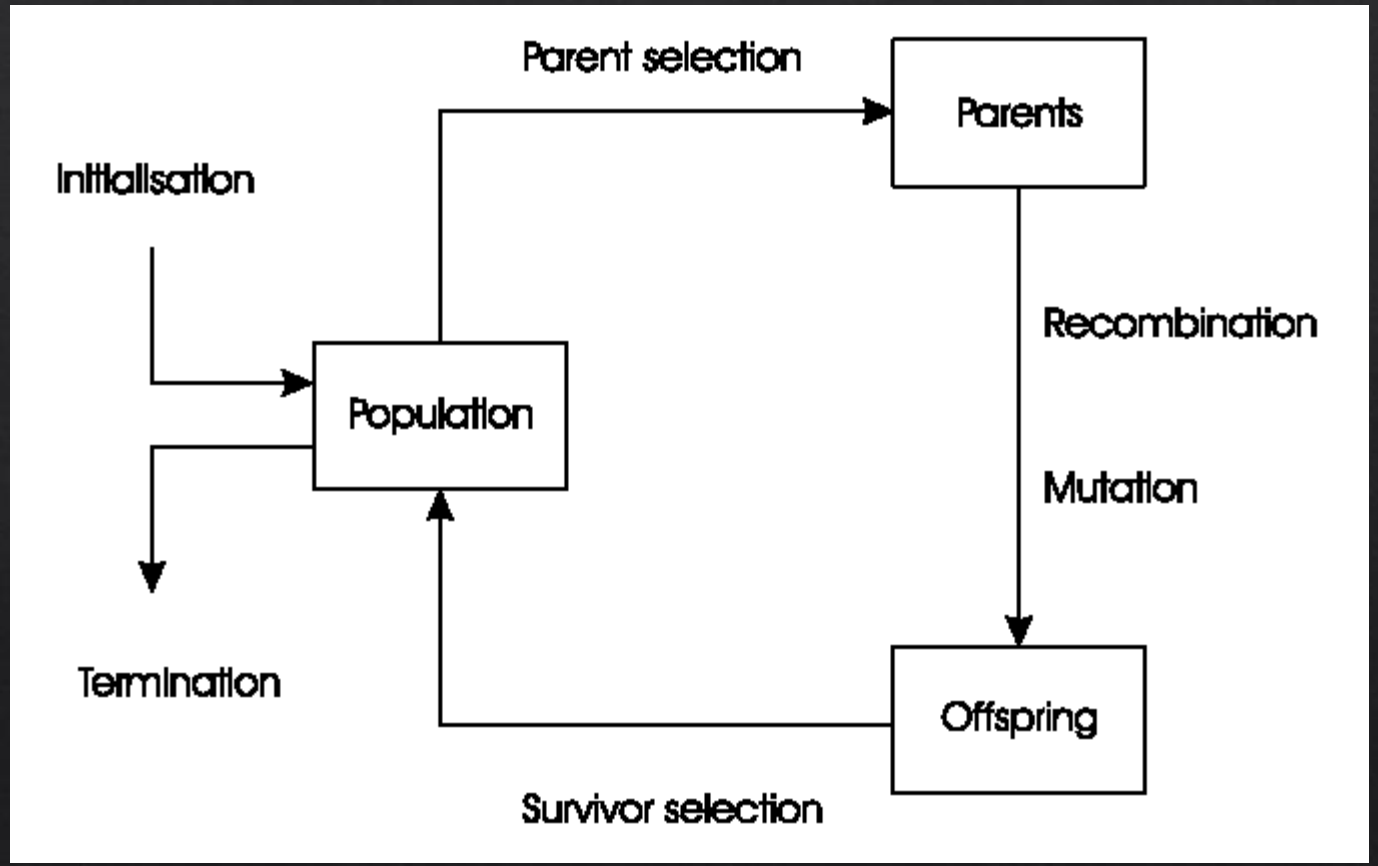
- Return best tour found at any time



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

Genetic Algorithms

General Scheme of GAs



Pseudo-code for typical GA

```
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END
```

Representations

- Candidate solutions (individuals) exist in *phenotype* space
- They are encoded in chromosomes, which exist in *genotype* space
  - Encoding : phenotype=> genotype
  - Decoding : genotype=> phenotype
- Chromosomes contain genes, which are in (usually fixed) positions called loci (sing. locus) and have a value (allele)
- In order to find the global optimum, every feasible solution must be represented in genotype space

Short version: You must be able to represent ALL solutions to the problem simply (say…as a string)

Evaluation (Fitness) Function

- Represents the requirements that the population should adapt to
- a.k.a. *quality* function or *objective* function
- Assigns a single real-valued fitness to each phenotype which forms the basis for selection
  - So the more discrimination (different values) the better
- Typically we talk about fitness being maximised
  - Some problems may be best posed as minimisation problems, but conversion is trivial

- **Short version: fitness function evaluates HOW GOOD a solution you've seen solves the given problem. Higher values mean the solution is closer to optimal.**

Population

- Holds (representations of) possible solutions
- Usually has a fixed size and is a *multi-set* of genotypes
- Selection operators usually take whole population into account i.e., reproductive probabilities are *relative* to *current* generation
- Diversity of a population refers to the number of different fitnesses / phenotypes / genotypes present (note not the same thing)

Parent Selection Mechanism

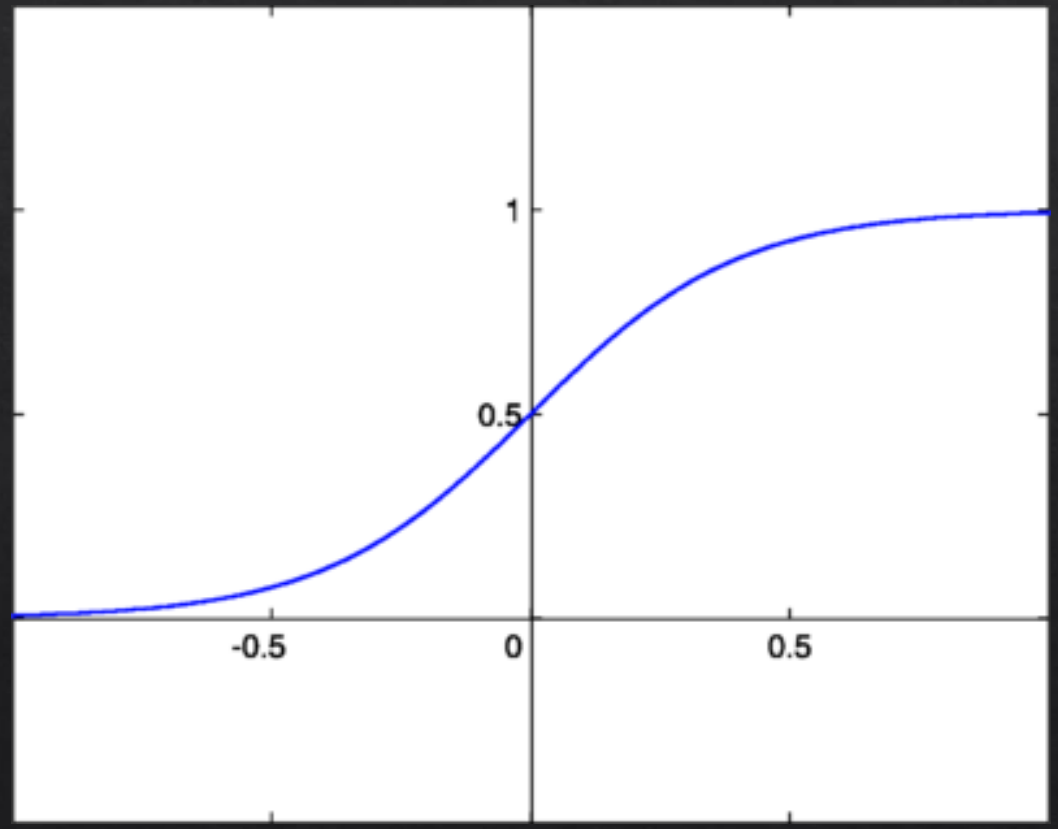
- Assigns variable probabilities of individuals acting as parents depending on their fitnesses
- Usually probabilistic
  - high quality solutions more likely to become parents than low quality
  - but not guaranteed
  - even worst in current population usually has non-zero probability of becoming a parent
- This *stochastic* nature can aid escape from local optima
  - What does this mean?? You should understand this idea. Only matters when search space is VERY large, otherwise we could just exhaustively search all possible offspring and return max fitness

Parent Selection Mechanism

- Use our friend, the sigmoid function.

- How would this work?
  - Need to map fitness quality onto the x-axis so that good solutions have high probabilities of being chosen as parents.

  - Then, flip some coins to see which parents get selected.



Variation Operators

- Role is to generate new candidate solutions

- Usually divided into two types according to their arity (number of inputs):
  - Arity 1 : mutation operators
  - Arity >1 : Recombination operators
  - Arity = 2 typically called crossover

Mutation

- Acts on one genotype and delivers another
- Element of randomness is essential and differentiates it from other unary heuristic operators
- Importance ascribed depends on representation and dialect:
  - Binary GAs – background operator responsible for preserving and introducing diversity
  - EP for FSM's/ continuous variables – only search operator
  - GP – hardly used
- May guarantee connectedness of search space and hence convergence proofs

Recombination

- Merges information from parents into offspring
- Choice of what information to merge is stochastic
- Most offspring may be worse, or the same as the parents
- Hope is that some are better by combining elements of genotypes that lead to good traits
- Principle has been used for millennia by breeders of plants and livestock

Survivor Selection

- a.k.a. ***replacement***
- Most EAs use fixed population size so need a way of going from (parents + offspring) to next generation
- Often deterministic
  - Fitness based : e.g., rank parents+offspring and take best
  - Age based: make as many offspring as parents and delete all parents
- Sometimes do combination (elitism)

Initialization / Termination

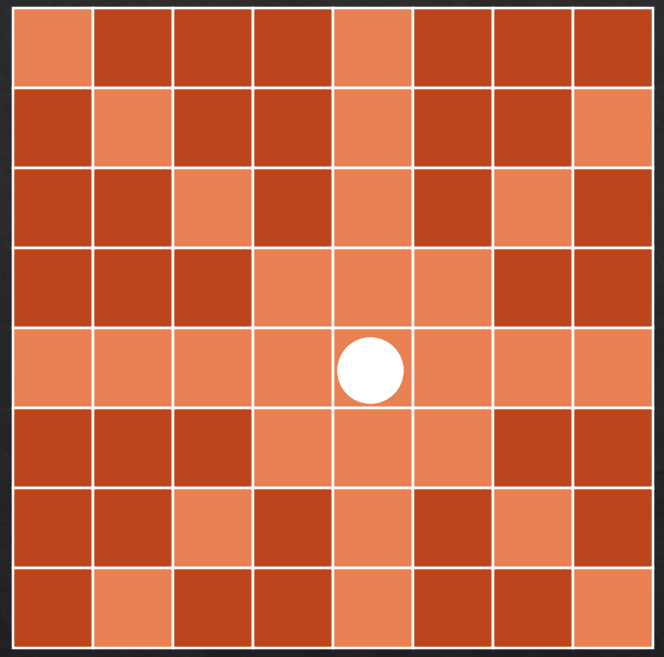
- Initialization usually done at random,
  - Need to ensure even spread and mixture of possible allele values
  - Can include existing solutions, or use problem-specific heuristics, to “seed” the population
- Termination condition checked every generation
  - Reaching some (known/hoped for) fitness
  - Reaching some maximum allowed number of generations
  - Reaching some minimum level of diversity
  - Reaching some specified number of generations without fitness improvement

Fun Example

Smart Rockets:

http://www.blprnt.com/smartrockets/

Example: the 8 queens problem



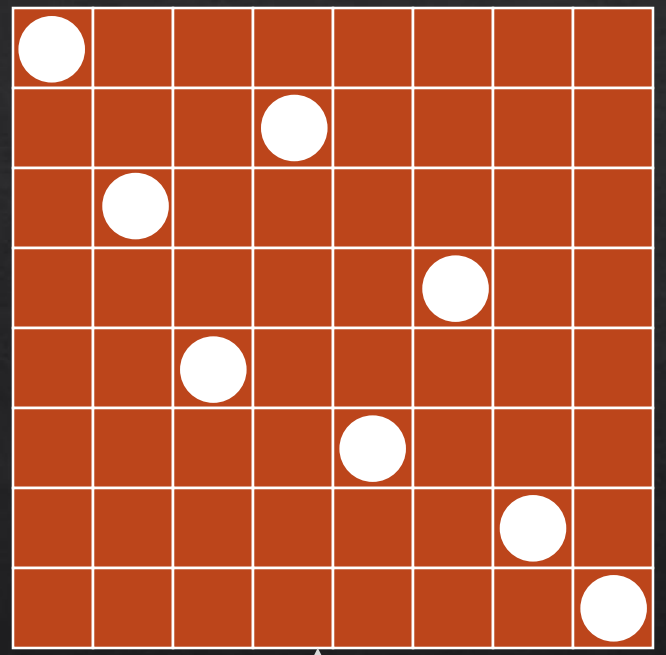
Place 8 queens on an 8x8 chessboard in
such a way that they cannot check each other

Harder when NxN board and need to place N queens

The 8 queens problem: representation

Phenotype:
a board configuration

Genotype:
a permutation of
the numbers 1 - 8



Obvious mapping

1	3	5	2	6	4	7	8

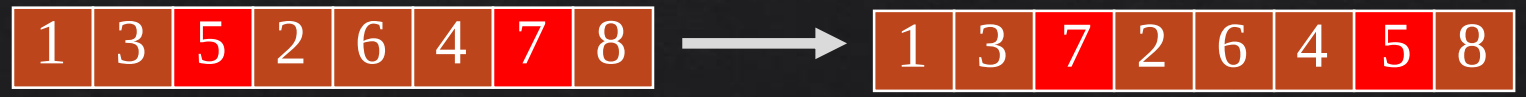
8 Queens Problem: Fitness evaluation

- Penalty of one queen:
  the number of queens she can check.
- Penalty of a configuration:
  the sum of the penalties of all queens.
- Note: penalty is to be minimized
- Fitness of a configuration:
  inverse penalty to be maximized

The 8 queens problem: Mutation

Small variation in one permutation, e.g.:

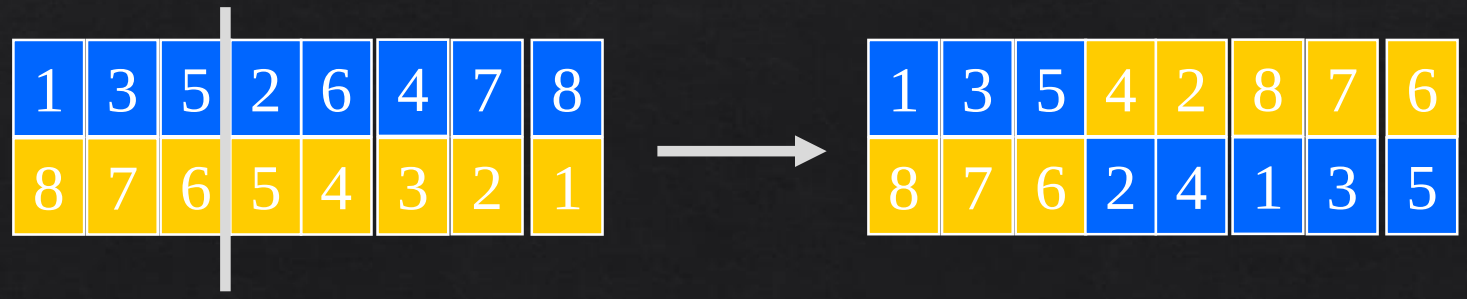
- swapping values of two randomly chosen positions,



The 8 queens problem: Recombination

Combining two permutations into two new permutations:

- choose random crossover point
- copy first parts into children
- create second part by inserting values from other parent:
  - in the order they appear there
  - beginning after crossover point
  - skipping values already in child



The 8 queens problem: Selection

- Parent selection:
  - Pick 5 parents and take best two to undergo crossover
- Survivor selection (replacement)
  - insert the two new children into the population
  - sort the whole population by decreasing fitness
  - delete the worst two

8 Queens Problem: summary

Representation	Permutations
Recombination	"Cut-and-crossfill" crossover
Recombination probability	100%
Mutation	Swap
Mutation probability	80%
Parent selection	Best 2 out of random 5
Survival selection	Replace worst
Population size	100
Number of Offspring	2
Initialisation	Random
Termination condition	Solution or 10,000 fitness evaluation

Note that this is ***only one possible***
set of choices of operators and parameters

Practice: 3-Coloring

- Do on your own:

- *Design a GA that attempts to 3-color a graph*