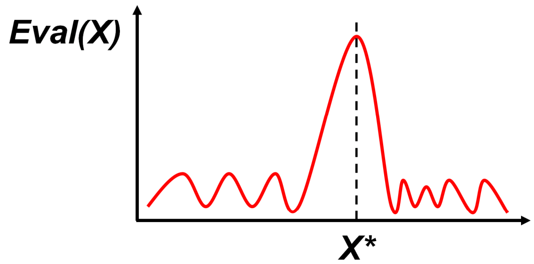
Lecture 4:
Local/Stochastic Search

Local Search

Given:

- **A set of states (configurations) $S = \{X_1,\ldots,X_M\}$**
- **A function that evaluates each state: *Eval*(X)**

Find global maximum: X^* such that *Eval*(X^*) is greater than all *Eval*(X_i) for all values of X_i

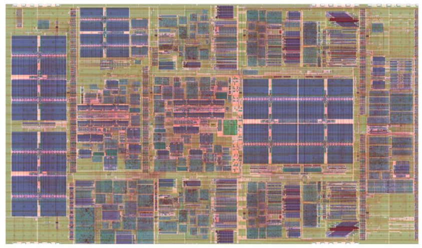


In This Lecture

- **Either set of configurations too large to be enumerated explicitly**
- **Or computation of *Eval*(.) may be expensive**
- **Therefore we cannot find the maximum of *Eval*(.) by simply trying out all states**
- **Solutions with similar values of *Eval*(.) are considered equivalent for the problem at hand**
- **We do not care how we get to X* (the path), we care only about the description of the configuration X***

Up until now we cared about the path. Now we don't. We're in charge, so we can do that.

Real-World Examples



- **VLSI layout:**
  - ***X* = placement of components + routing of interconnections**
  - ***Eval* = Distance between components + %unused + routing length + ?**

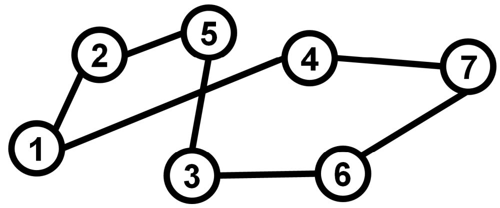
Too many configurations to look at them all

Once we find the best configuration, we don't really care about the path to the solution.

Real-World Examples

- **Scheduling: Given *m* machines, *n* jobs**
- ***X* = assignment of jobs to machines**
- ***Eval* = completion time of the *n* jobs (minimize)**

Example: TSP (Traveling Salesperson)



X_1 = {1 2 5 3 6 7 4}

- **Configuration *X* = Path through all the nodes**
- ***Eval* = Length of path**
- **Size of search space = (*N*-1)!/2**

TSP: You, the salesperson, wants to travel through every city once in the cheapest way possible.

(N-1)!/2 is correct because you don't care about the starting city (hence (N-1)! instead of N!), and you don't care about the direction (hence the /2)

Example: SAT (SATisfiability)

$\mathbf{A} \vee \neg\mathbf{B} \vee \mathbf{C}$ clause

$\neg\mathbf{A} \vee \mathbf{C} \vee \mathbf{D}$

$\mathbf{B} \vee \mathbf{D} \vee \neg\mathbf{E}$

$\neg\mathbf{C} \vee \neg\mathbf{D} \vee \neg\mathbf{E}$

literal

$\neg\mathbf{A} \vee \neg\mathbf{C} \vee \mathbf{E}$

	A	**B**	**C**	**D**	**E**	*Eval*
X_1	*true*	*true*	*false*	*true*	*false*	**5**
X_2	*true*	*true*	*true*	*true*	*true*	**4**

Configuration: Assignment of true/false to each variable

Eval: Number of clauses satisfied

Real world examples of SAT problems?

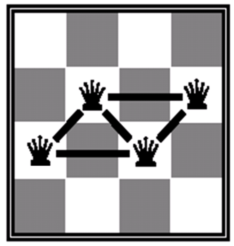
Model checking

Mine sweeper

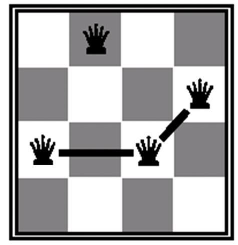
Sudoku

College class scheduling

Example: N-Queens



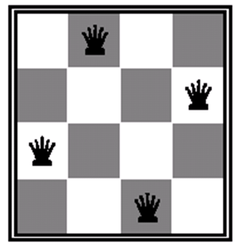
***Eval*(*X*) = 5**



***Eval*(*X*) = 2**

Find a configuration in which no queen can attack any other queen

What's Eval() here?



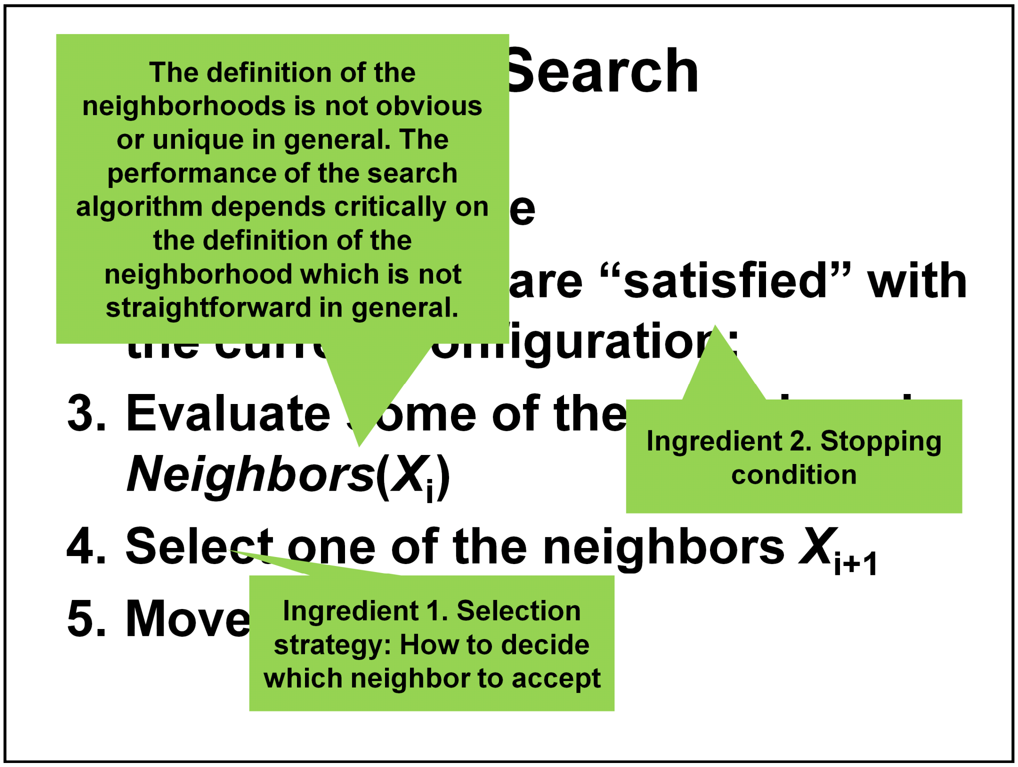
***Eval*(*X*) = 0**

N^N configurations (One queen per column)

Eval function is number of ways a queen can attack another queen

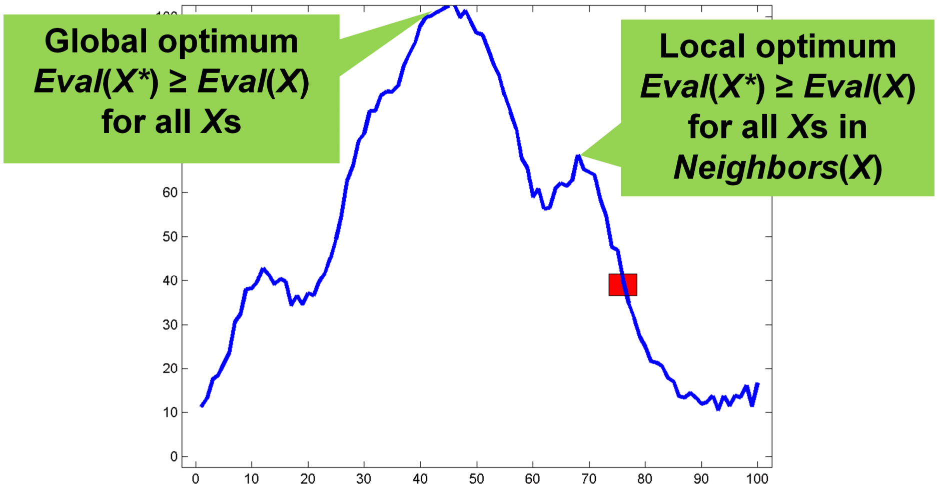
Local Search

1. X_o , ← Initial state
2. Repeat until we are "satisfied" with the current configuration:
3. Evaluate some of the neighbors in *Neighbors*(X_i)
4. Select one of the neighbors X_{i+1}
5. Move to X_{i+1}



Lots of questions to answer

Simplest Example



$$S = \{1,\ldots,100\}$$

$$Neighbors(X) = \{X\text{-}1, X\text{+}1\}$$

Start at red rectangly thing

Look at your left neighbor X-1, and your right neighbor X+1 and go whichever way is better.

You get stuck in a local optimum, though. Phooey

Most Basic Algorithm: Hill-Climbing (Greedy Local Search)

- X ← Initial configuration
- Iterate:
  1. $E \leftarrow Eval(X)$
  2. $\mathcal{N} \leftarrow Neighbors(X)$
  3. For each X_i in $\mathcal{N}$
     $E_i \leftarrow Eval(X_i)$
  4. If all E_i's are lower than E
     Return X
     Else
     $i^* = \text{argmax}_i (E_i)$ $X \leftarrow X_{i^*}$ $E \leftarrow E_{i^*}$

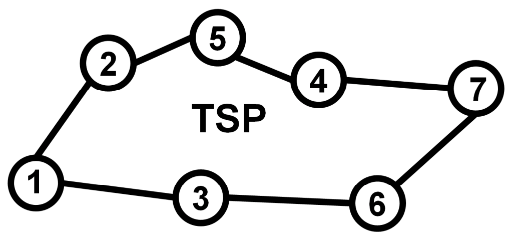
Called hill climbing because you always try to go up hill.

Neighbors could be more than just one left and one right. You could look much further.

Problem, however, is getting stuck in local optimum

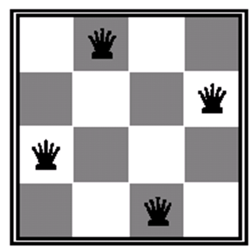
More Interesting Examples

How can we define *Neighbors*(*X*)?



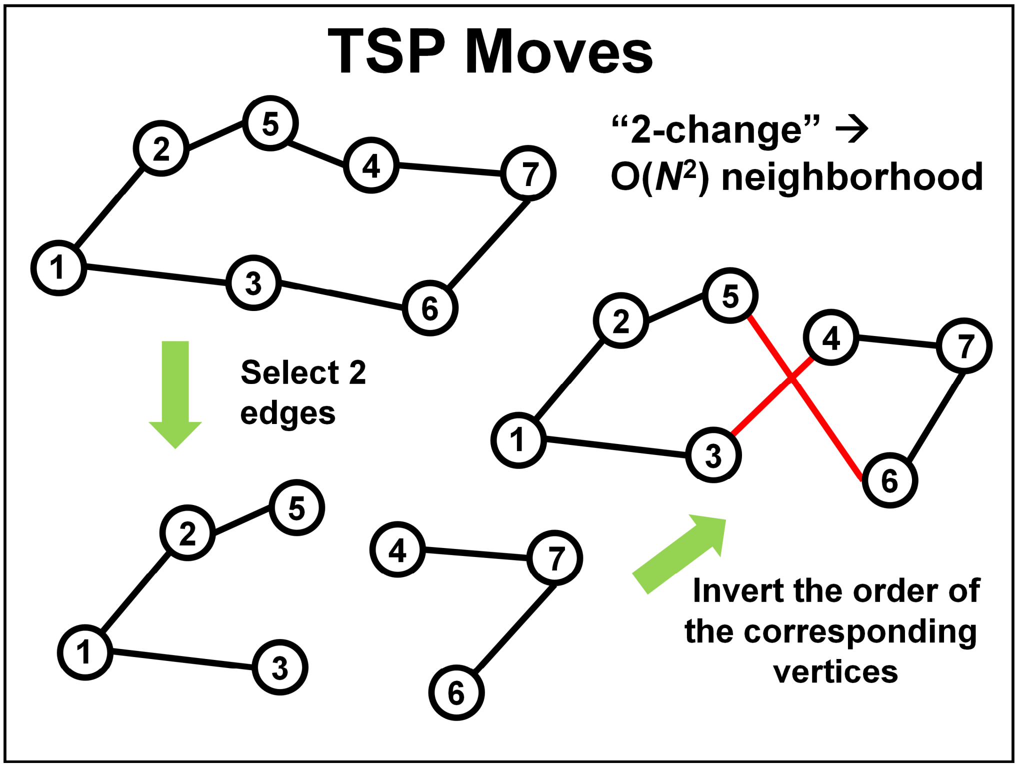
SAT
$A \lor \neg B \lor C$
$\neg A \lor C \lor D$
$B \lor D \lor \neg E$
$\neg C \lor \neg D \lor \neg E$
$\neg A \lor \neg C \lor E$

N-Queens

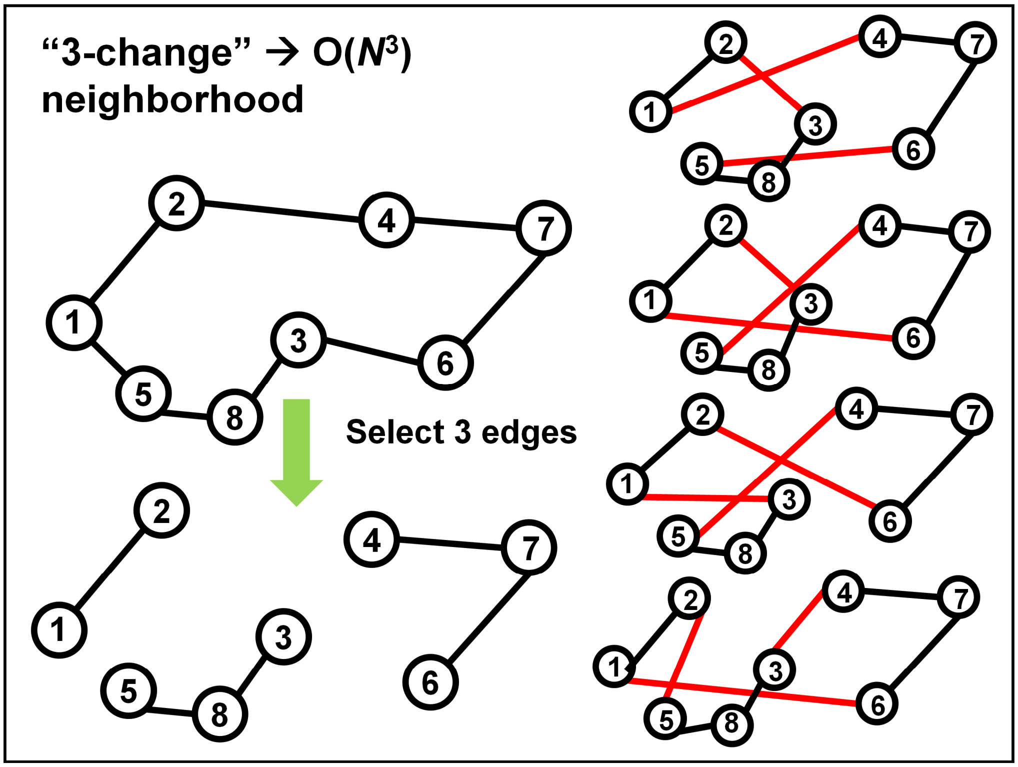


For SAT, we could say that neighbors are configurations with one variable changed.

For TSP, we could say that neighbors are configurations with two edges swapped.



O(N^2) neighbors, because there are (n choose 2) possible pairs of edges to swap



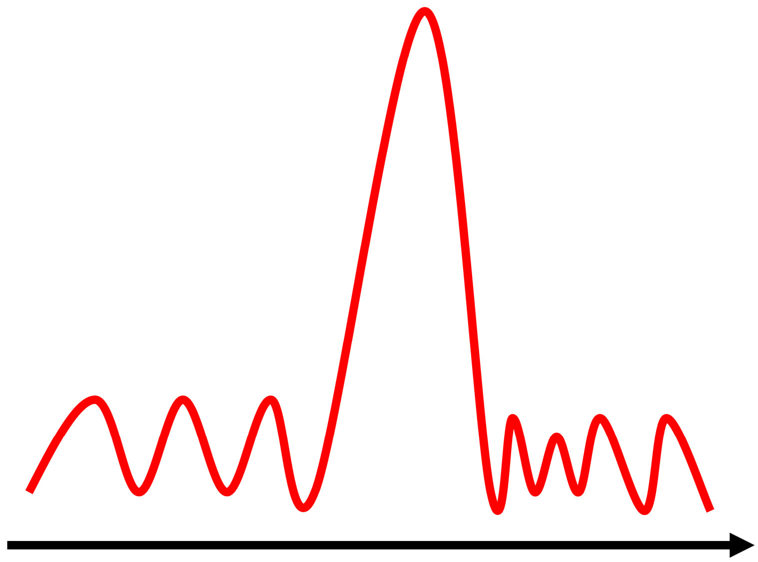
n choose 3 is O(N^3)

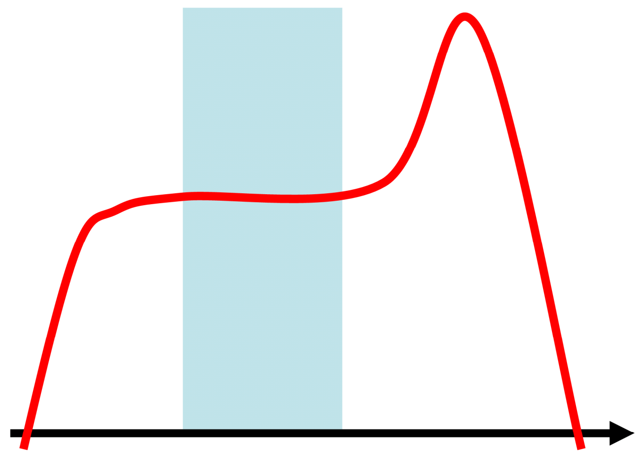
Issues

Trade-off on size of neighborhood:

- **Larger neighborhood = better chance of finding a good maximum but may require evaluating an enormous number of moves**
- **Smaller neighborhood = smaller number of evaluations but may get stuck in poor local maxima**

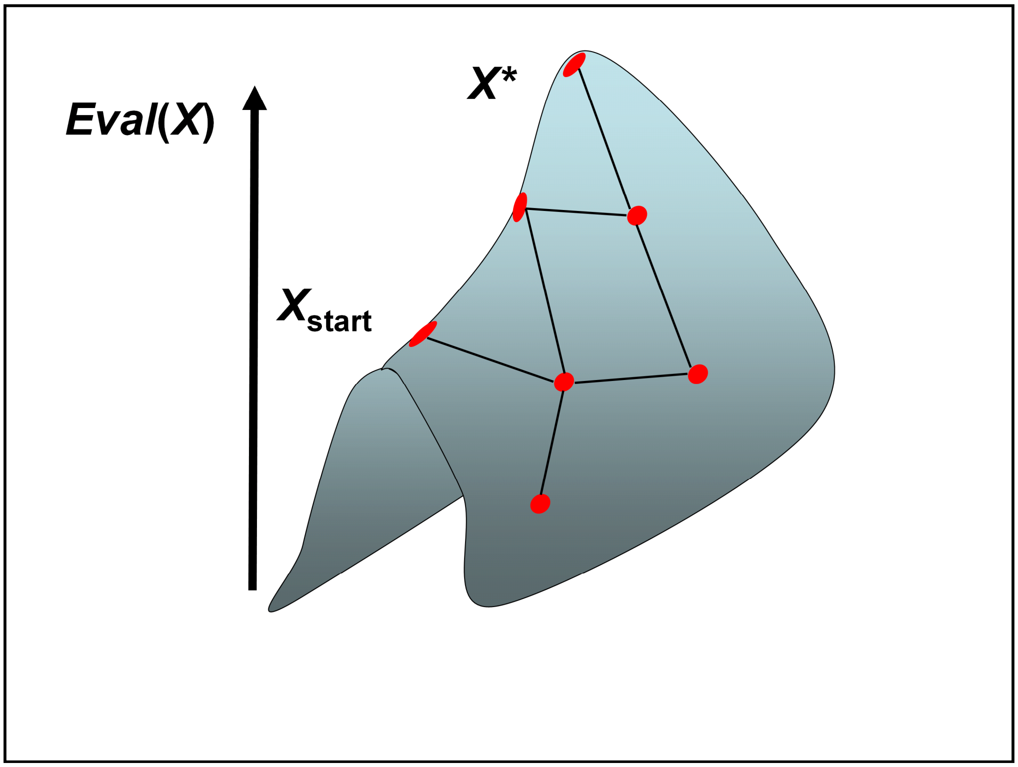
Multiple "Poor" Local Maxima





Plateau = constant region of *Eval*(.)

No one likes plateaus



To get from X_{start} to X* you need to first go down…so X_{start} is a local optimum

Remarks

- How much memory is used?
- All we can hope is to find the local maximum “closest” to the initial configuration. Can we do better than that?

Memory used: very little...constant amount…pretty much the size of the neighborhood

Stochastic Search: Randomized Hill-Climbing

- ***X*** **← Initial configuration**
- **Iterate:**

Until when?

1. $E \leftarrow Eval(X)$
2. $X' \leftarrow$ one configuration randomly selected in $Neighbors(X)$
3. $E' \leftarrow Eval(X')$
4. If $E' > E$

   $X \leftarrow X'$

   $E \leftarrow E'$

Critical change: We no longer select the best move in the entire neighborhood

WALKSAT

Iterate until all clauses are satisfied or max iterations:

1. **Select an unsatisfied clause**
2. **With probability *p*:**
   Select a variable x_i at random
3. **With probability 1-*p*:**
   Select the variable x_i such that changing x_i will unsatisfy the least number of clauses (Max of *Eval*(*X*))
4. **Change the assignment of the selected variable x_i**

Random walk part

Greedy part

One of the best algorithms for SAT

In this algorithm, you don't always go uphill. When you randomly pick a variable, it could cause you to go downhill!

X_i is a variable in the unsatisfied clause

Simulated Annealing

1. $E \leftarrow Eval(X)$
2. $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)
3. $E' \leftarrow Eval(X')$
4. If $E' \geq E$

   $X \leftarrow X'$

   $E \leftarrow E'$

   Else accept the move to X' with some probability p:

   $X \leftarrow X'$

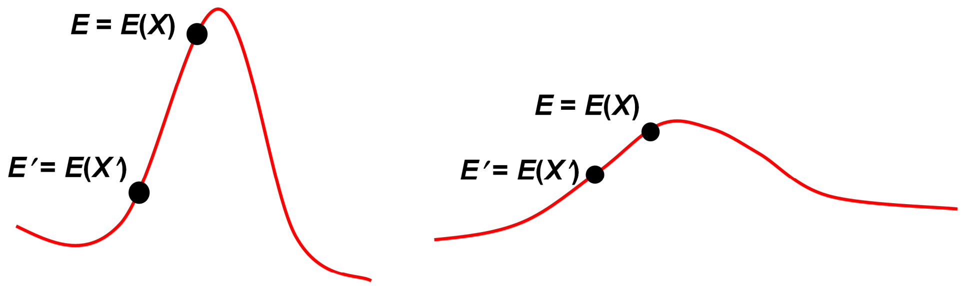
   $E \leftarrow E'$

Critical change: We no longer move always uphill. Next question: How to choose *p*?

From Wikipedia (http://en.wikipedia.org/wiki/Simulated_annealing):

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

How to set *p*? Intuition



E* – *E′ **is large: It is more likely that we are moving toward a (promising) sharp maximum so we don't want to move downhill too much**

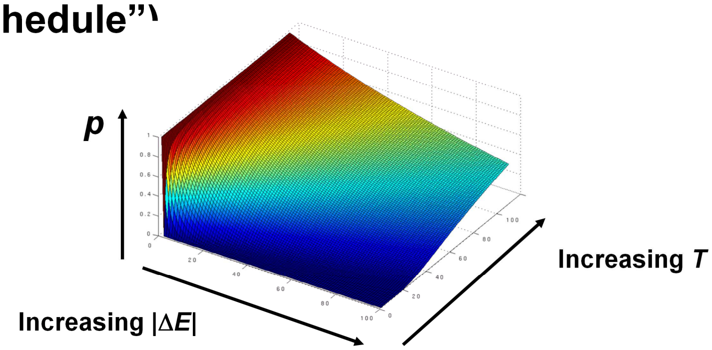
E* – *E′ **is small: It is likely that we are moving toward a shallow maximum that is likely to be a (uninteresting) local maximum, so we like to move downhill to explore other parts of the landscape**

Choosing *p*: Simulated Annealing

- If $E' \geq E$ accept the move
- Else accept the move with probability:

$$p = e^{-(E - E')/T}$$

- Start with high temperature T and decrease T gradually as iterations increase ("cooling schedule")



This temperature stuff is they metaphor-y part.

Simulated Annealing

X **← Initial configuration**

T **← Initial high temperature**

Iterate:

1. **Do *K* times:**

   $E \leftarrow Eval(X)$

   $X' \leftarrow$ **one configuration randomly selected in** ***Neighbors*** **(*X*)**

   $E' \leftarrow Eval(X')$

   If $E' \geq E$

   $X \leftarrow X'; E \leftarrow E';$

   Else accept the move with prob $p = e^{-(E - E')/T}$**:**

   $X \leftarrow X'; E \leftarrow E';$

2. $T \leftarrow \alpha T$ ∞

alpha < 1

As t goes down, probability p goes down. And as p goes down you take fewer chances.

Stopping condition depends on the problem.

Simulated Annealing

X ← Initial configuration

T ← Initial high temperature

Iterate:

1. Do *K* times:

    E ← *Eval*(*X*)

    X' ← one configuration randomly selected in *Neighbors* (*X*)

    E' ←

    If *E*

    X … ← *E'*;

    Else acc… the move with prob $p = e^{-(E - E')/T}$:

    X ← *X'*; *E* ←

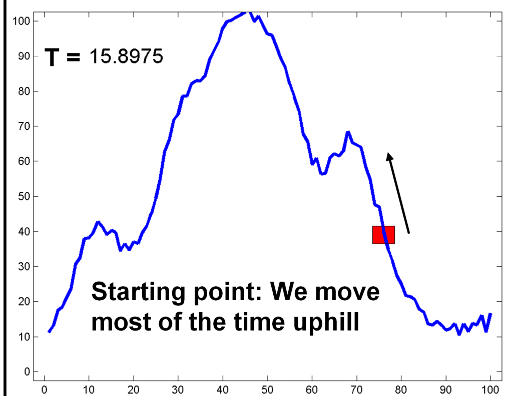
2. $T \leftarrow \alpha T$

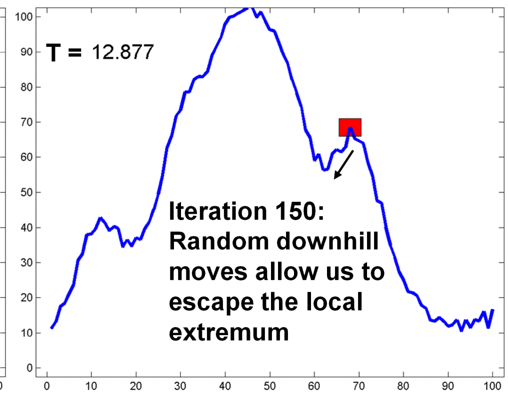
Iterate a number of times keeping the temperature fixed

Progressively decrease the temperature using an exponential cooling schedule:
$T(n) = \alpha^n T$ with $\alpha < 1$

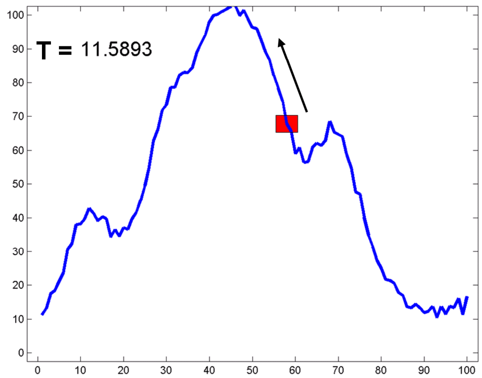
$T = 0$ → Greedy hill climbing
$T = \infty$ → Random walk

Basic Example

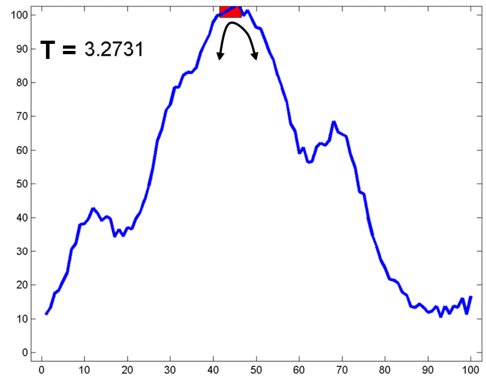




Basic Example

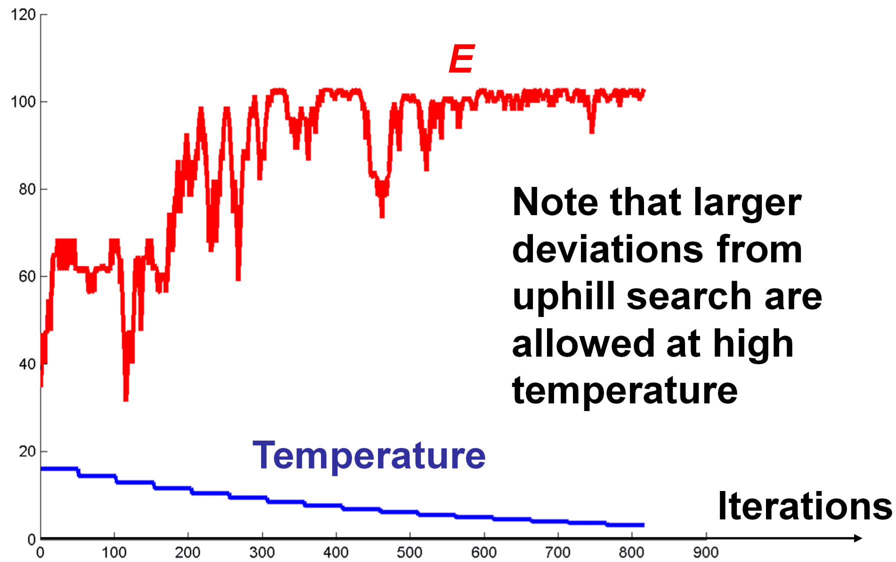


Iteration 180: Random downhill moves have pushed us past the local extremum



Iteration 800: As T decreases, fewer downhill moves are allowed and we stay at the maximum

Basic Example



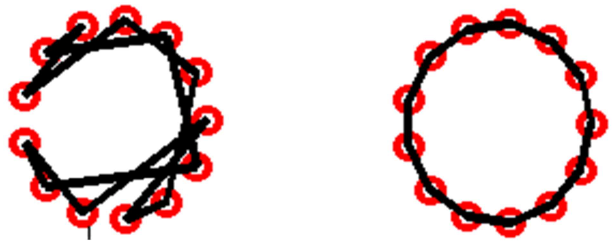
Where does this come from?

- **If the temperature of a solid is *T*, the probability of moving between two states of energy is:**

$$e^{-\Delta Energy/kT}$$

- **If the temperature *T* of a solid is decreased slowly, it will reach an equilibrium at which the probability of the solid being in a particular state is:**
- ***Probability* (*State*) proportional to** $e^{-Energy(State)/kT}$
- **Boltzmann distribution → States of low energy relative to *T* are more likely**
- **Analogy:**
  - **State of solid ←→ Configurations *X***
  - **Energy ←→ Evaluation function Eval(.)**

A TSP Example



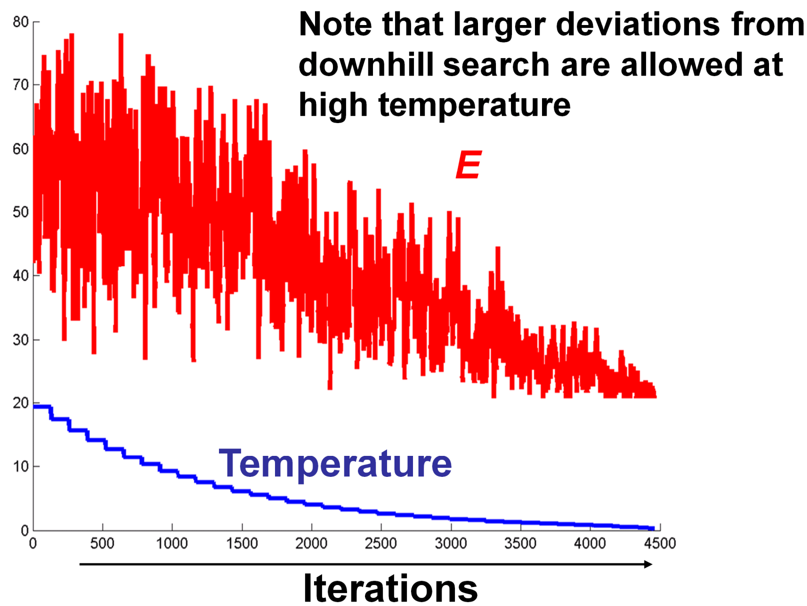
N = 13 nodes (in a circle)

Repeat K = 100N times

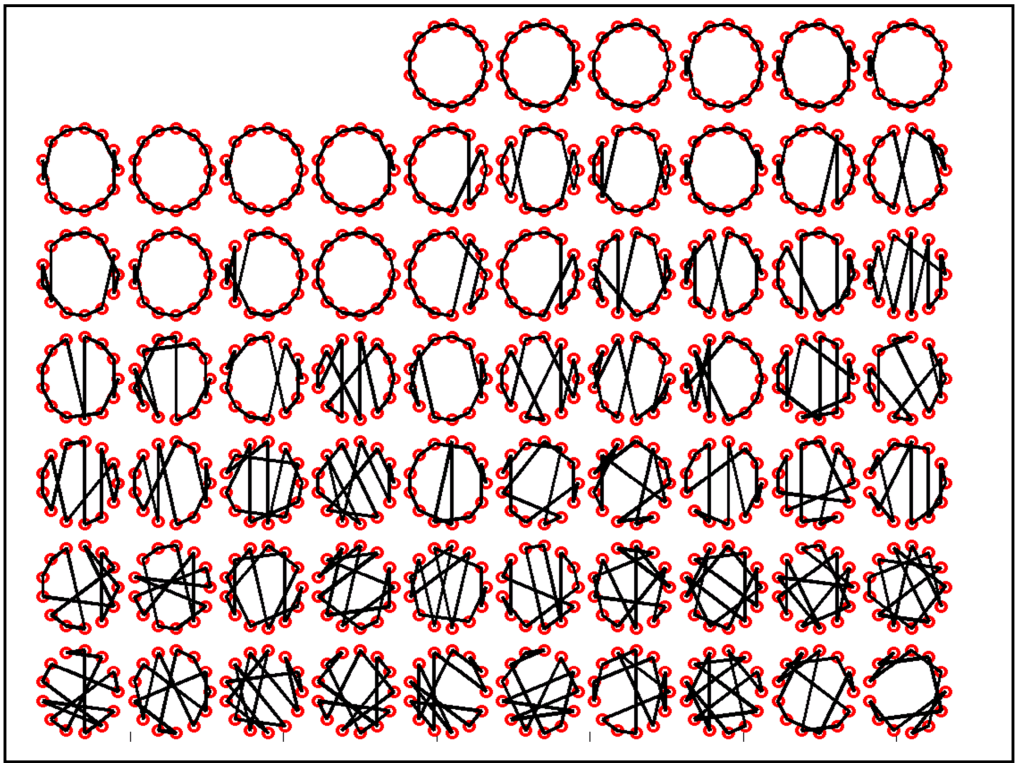
Optimal configuration has E = 25

Starting configuration has E = 55

A TSP Example

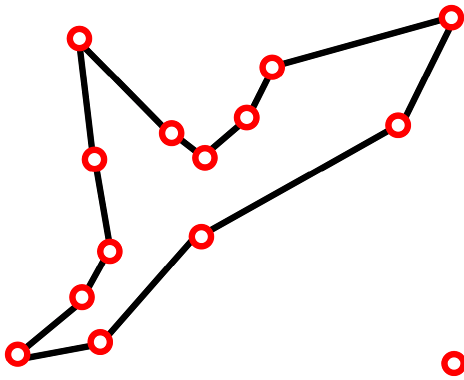


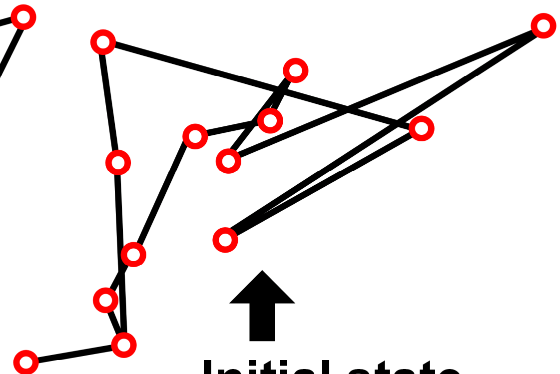
Energy of the configurations as the algorithm runs



Start bottom right, end top left.

Another Example



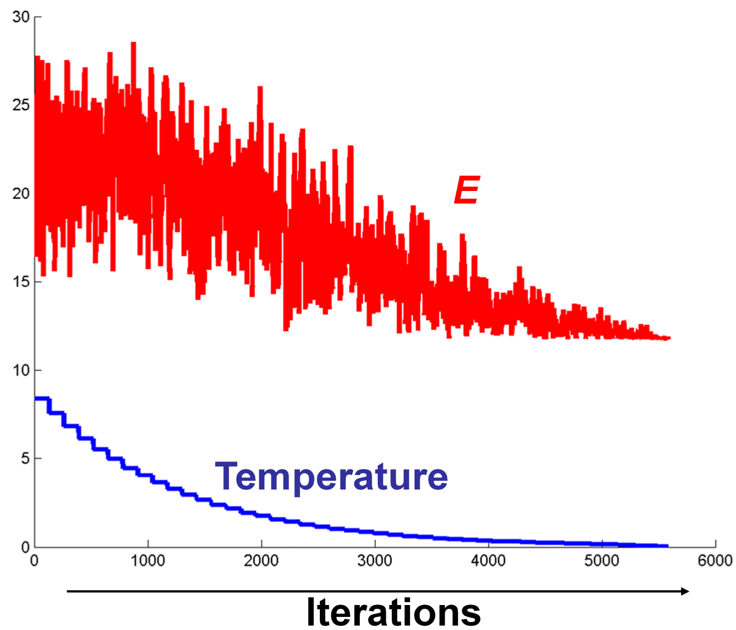


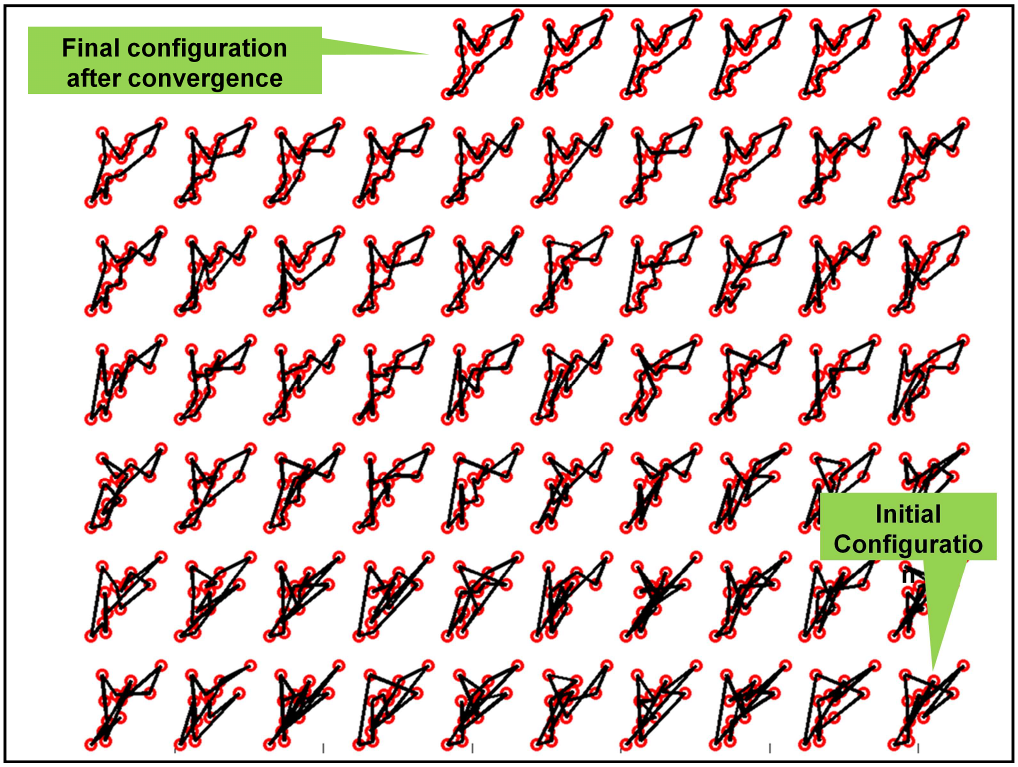
Initial state

N = 13 nodes

Repeat K = 100N times

Another Example




Final configuration after convergence
Initial Configuratio
n

What can we say about convergence?

- **In theory:**

$$\lim_{T \to 0} \lim_{K \to \infty} \Pr(X(T, K) \in S^*) = 1$$

In words: Probability that the state reached after *K* iterations at temperature *T* is a global optimum

- **In practice:**
  - **Perform a large enough number of iterations (*K* "large enough")**
  - **Decrease temperature slowly enough (α "close enough" to 1)**
  - **But, if not careful, we may have to perform an enormous number of evaluations**

"Belongs to S*" just means an optimal solution

So...if you run an infinite number of iterations you will eventually find the optimal solution. Note…the limit as T goes to 0 does NOT mean T = 0.

Simulated annealing is a useful algorithm that is actually used in real life, unlike most things Luis talks about

Simulated Annealing

$X \leftarrow$ Initial configuration

$T \leftarrow$ Initial high temperature

Iterate:

Many parameters need to be tweaked!!

1. Do K times:

   $E \leftarrow Eval(X)$

   $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)

   $E' \leftarrow Eval(X')$

   If $E' \geq E$

   $X \leftarrow X'; E \leftarrow E';$

   Else accept the move with prob $p = e^{-(E - E')/T}$:

   $X \leftarrow X'; E \leftarrow E';$

2. $T \leftarrow \alpha\, T$

Genetic/Evolutionary Algorithms

Genetic Algorithms

Configurations = Individuals in a population

Eval = measure of fitness

Least fit individuals DIE without reproducing

Most fit individuals reproduce more often

Each generation should be better than the past!

GA: Implementation

- Configurations represented by strings:

$$X = \boxed{1\,|\,0\,|\,0\,|\,1\,|\,1\,|\,0\,|\,0\,|\,1}$$

- Analogy:
  - The string is the chromosome representing the individual
  - String made up of genes
  - Configuration of genes are passed on to offsprings
  - Configurations of genes that contribute to high fitness tend to survive in the population
- Start with a random population of *P* configurations and apply two operations
  - *Reproduction*: Choose 2 "parents" and produce 2 "offsprings"
  - *Mutation*: Choose a random entry in one (randomly selected) configuration and change it

Genes are contiguous groups of 1's and 0's

Genetic Algorithms: Reproduction

Parents:

1	0	0	1	1	0	0	1

1	0	1	1	0	0	0	1

Genetic Algorithms: Reproduction

Parents:

1	0	0	1	1	0	0	1

1	0	1	1	0	0	0	1

Select random crossover point:

1	0	0	1	1	0	0	1

1	0	1	1	0	0	0	1

Genetic Algorithms: Reproduction

Parents:

1	0	0	1	1	0	0	1

1	0	1	1	0	0	0	1

Select random crossover point:

1	0	0	1	1	0	0	1

1	0	1	1	0	0	0	1

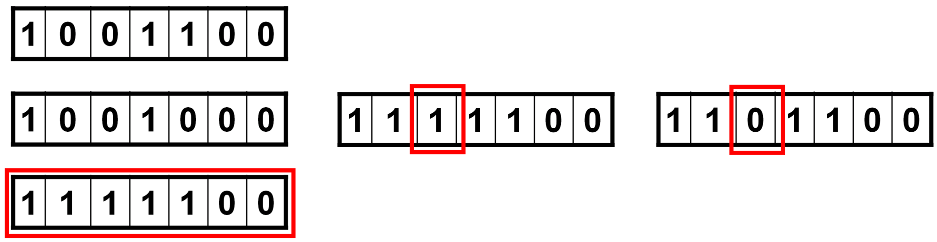
Offsprings:

1	0	0	1	0	0	0	1

1	0	1	1	1	0	0	1

An offspring receives part of the genes from each of the parents

Genetic Algorithms: Mutation



Select a random individual

Select a random entry

Change that entry

- Implements random deviations from inherited traits
- Corresponds loosely to “random walk”: Introduce random moves to avoid local extrema

Basic GA Outline

- Create initial population $X = \{X_1,..,X_P\}$
- Iterate:
  1. Select K random pairs of parents (X,X')
  2. For each pair of parents (X,X'):
     - 1.1 Generate offsprings (Y_1,Y_2) using crossover operation
     - 1.2 For each offspring Y_i:
       - Replace randomly selected element of the population by Y_i
       - With probability μ:
         - Apply a random mutation to Y_i
- Return the best individual in the population

Basic GA Outline

- Create initia ,X_P}
- Iterate:

Stopping condition is not obvious?

1. Select K random pairs of parents (X,X')
2. For each pair of arents

   1.1 Generate offspr

   ver operation

   ach offspring Y

   ce randomly s

   the population by Y

   With probability μ:

   Apply a random mutation to Y_i

- Return the best individual in the population

Variation: Generate only one offspring

Possible strategy: Select the best *rP* individuals (*r* < 1) for reproduction and discard the rest → Implements selection of the fittest

Genetic Algorithms: Selection

- **Discard the least-fit individuals through threshold on *Eval* or fixed percentage of population**
- **Select best-fit (larger *Eval*) parents in priority**
- **Example: Random selection of individual based on the probability distribution**

$$\Pr(\text{individual } X \text{ selected}) = \frac{Eval(X)}{\sum_{Y \in population} Eval(Y)}$$

- **Corresponds loosely to the greedy part of hill–climbing (we try to move uphill)**

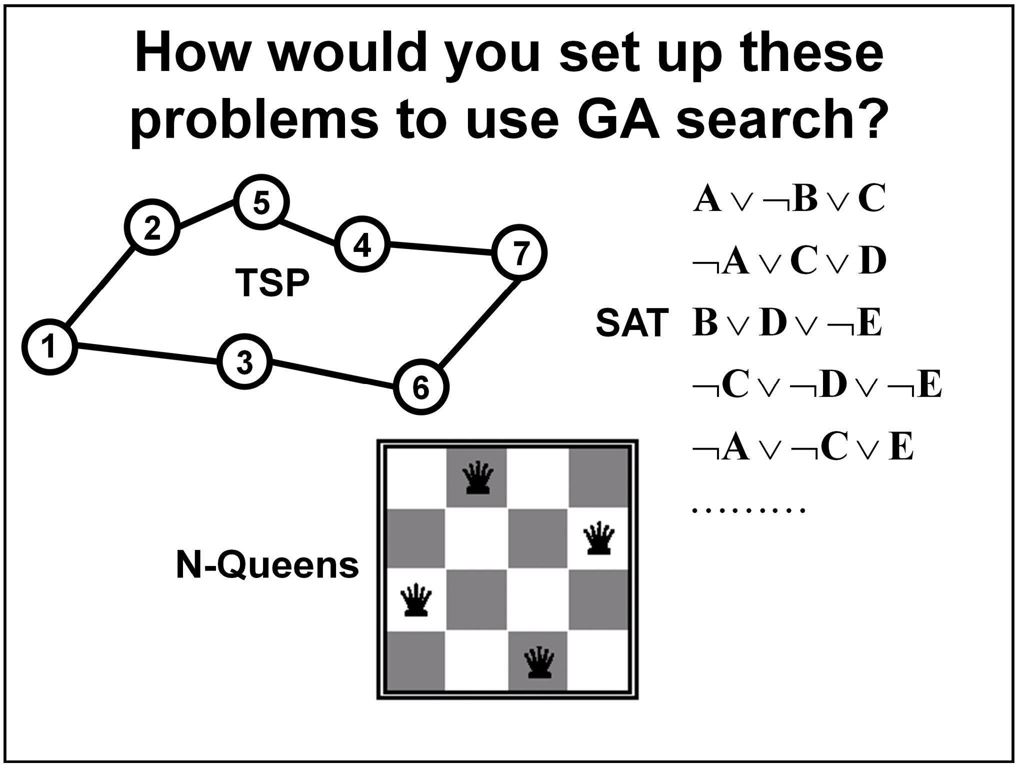
Ways to do selection

Basic GA Outline

- Create initial population $X = \{X_1,..,X_P\}$
- Iterate:
  1. Select K random
  2. For each pair of parents (X,X'):
     - 1.1 Generate offsprings (Y_1,Y_2) using crossover operation
     - ffspring Y_i:
     - ndomly selected element of
     - tion by Y_i
     - With probability μ:
       - Apply a random mutation to Y_i
- Return the best individual in the population

Hill-climbing component: Try to move uphill as much as possible

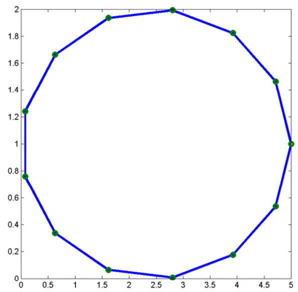
Random walk component: Move randomly to escape shallow local maxima



How to encode configurations as strings of 1’s and 0’s?

You want things that should stick together to be next to each other in the string

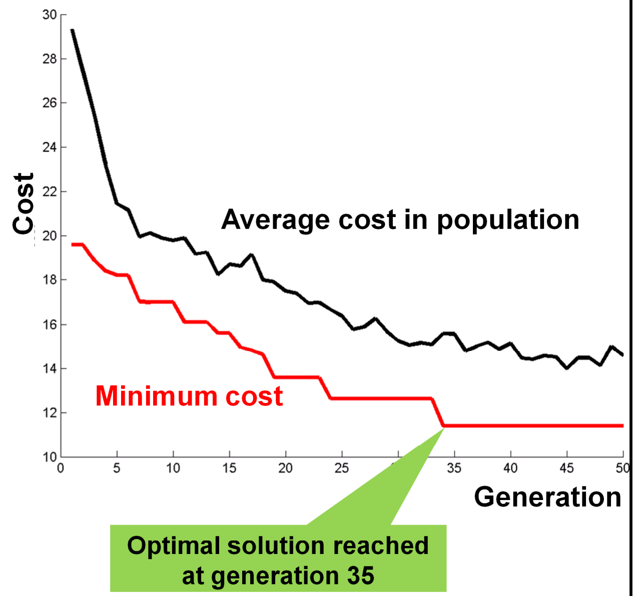
TSP Example

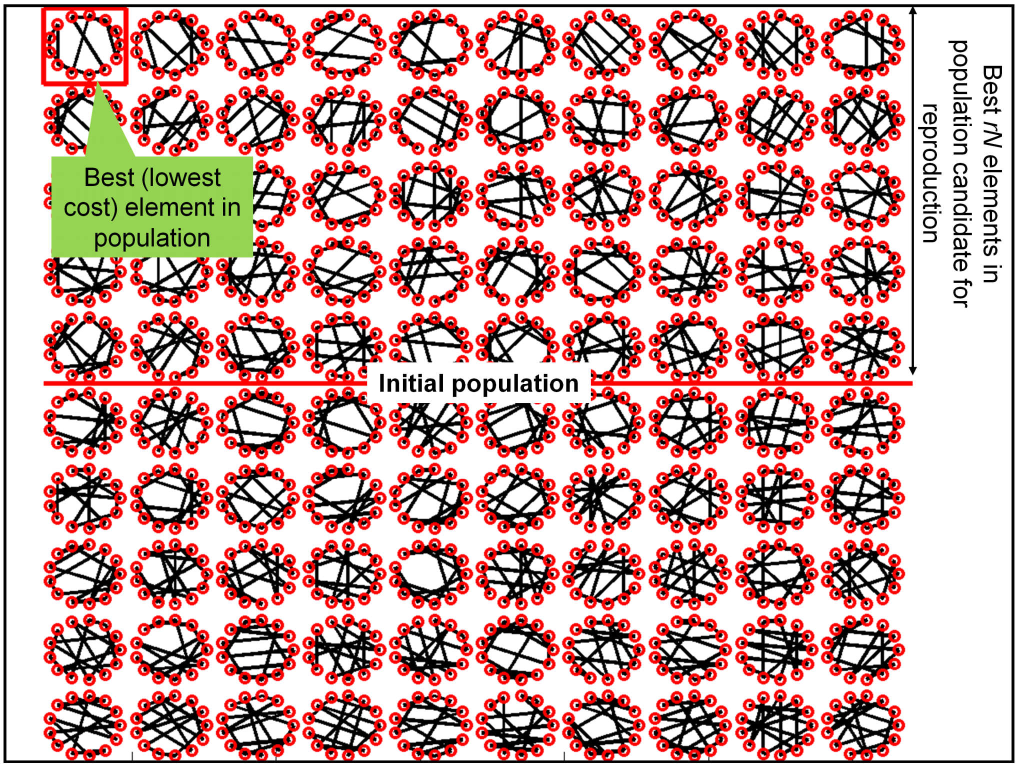


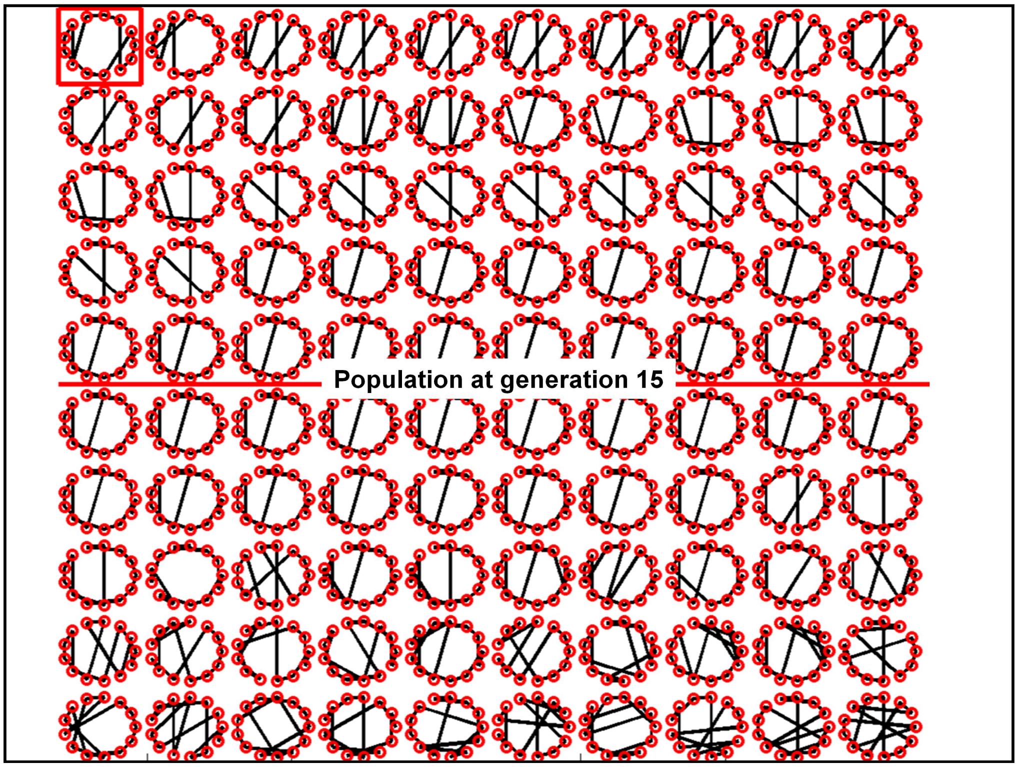
N **= 13**

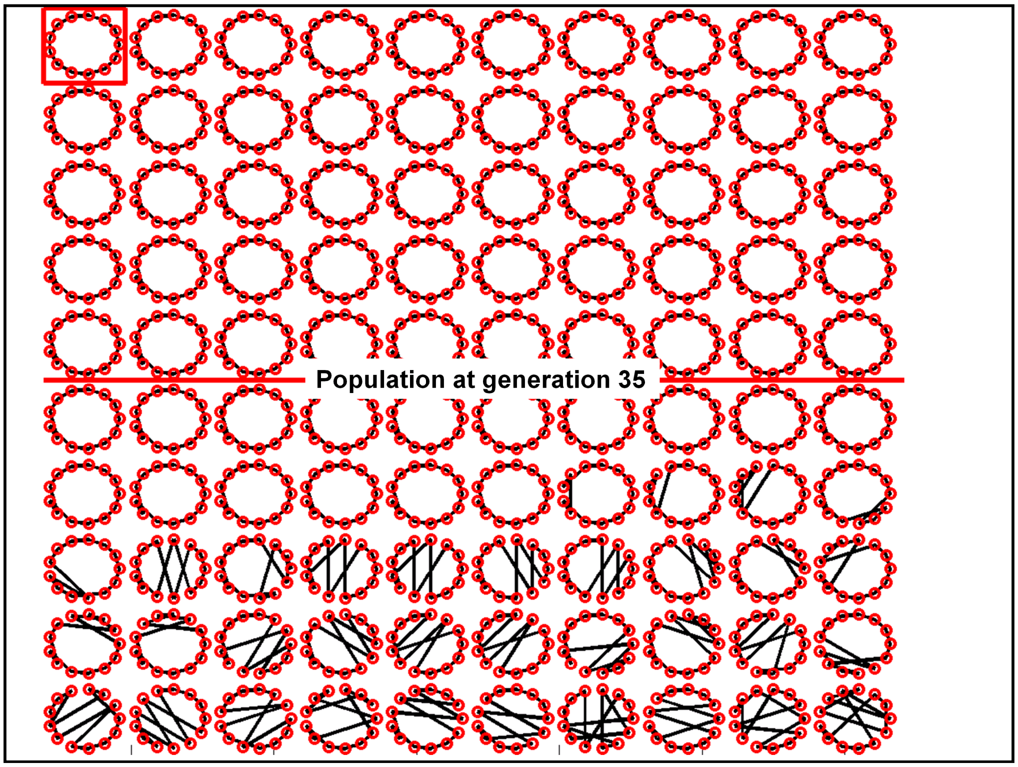
P **= 100 elements in population**

μ = 4% mutation rate
r **= 50% reproduction rate**

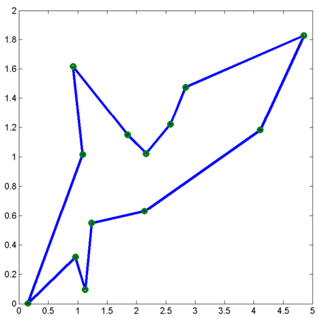



Best (lowest cost) element in population
Initial population
Best rN elements in population candidate for reproduction


Population at generation 15


Population at generation 35

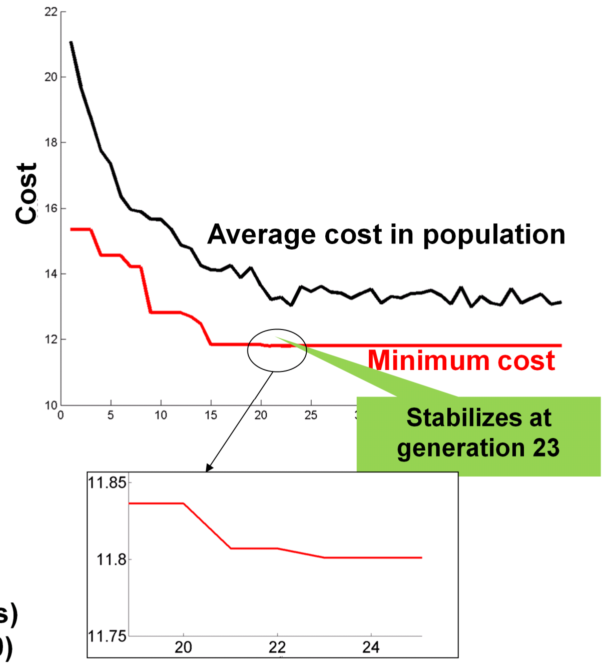
Another TSP Example

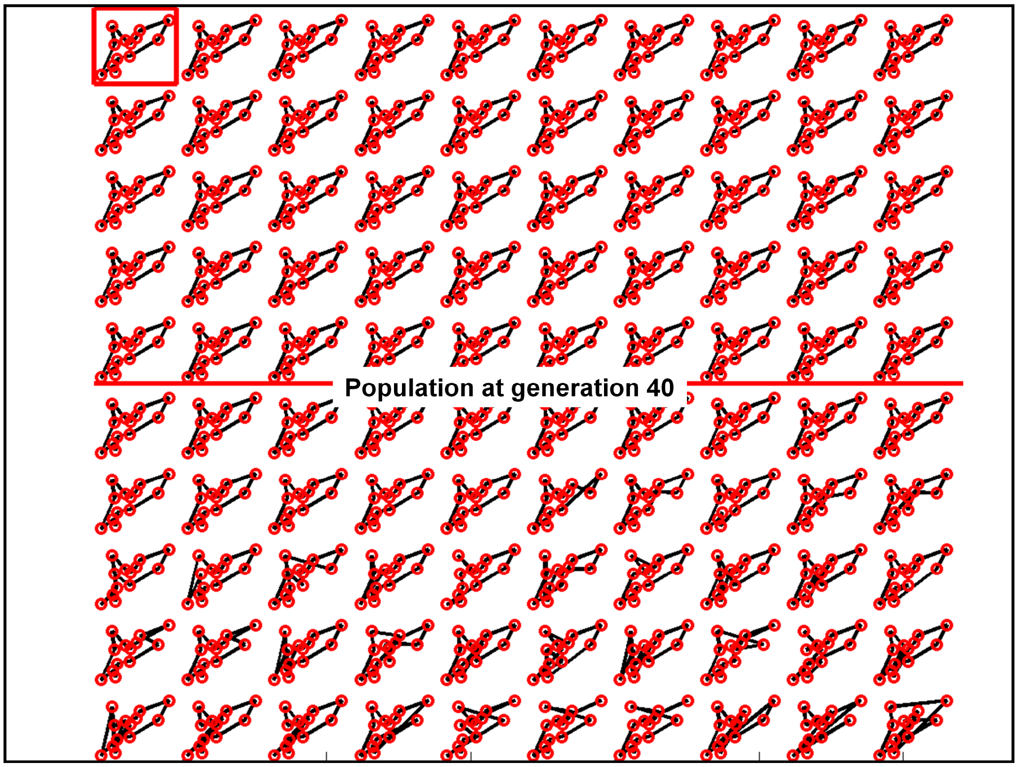


Converges and remains stable after generation 23

0.4% difference:
GA = 11.801
SA = 11.751

But: Number of operations (number of cost evaluations) much smaller (approx. 2500)




Population at generation 40

GA Discussion

- **Many parameters to tweak: μ, *P*, *r***
- **Many variations on basic scheme. Examples:**
  - **Multiple-point crossover**
  - **Dynamic encoding**
  - **Selection based on rank or relative fitness to least fit individual**
  - **Multiple fitness functions**
  - **Combine with a local optimizer (for example, local hill-climbing) → Deviates from "pure" evolutionary view**
- **In many problems, assuming correct choice of parameters, can be surprisingly effective**

People were extremely excited 20 years ago…not so much now…GA don't work very well. But they are cool.

Encoding a problem for a GA is very difficult to do well.

Here's a quote from Russell + Norvig:
"[It] is not clear whether the appeal of genetic algorithms arises from their performance or from their aesthetically pleasing origins in the theory of evolution."

GA Discussion

- **Why does it work at all?**
- **Limited theoretical results (informally!):**
  - **Suppose that there exists a partial assignment of genes *s* such that:**

$$\underset{X \text{ contains } s}{\text{Average of }} Eval(X) \geq \underset{Y \in \text{Population}}{\text{Average of }} Eval(Y)$$

  - **Then the number of individuals containing *s* will increase in the next generation**
- **Key consequence: The design of the representation (the chromosomes) is critical to the performance the GA. It is probably more important than the choice of parameters of selection strategy, etc.**

Summary

- **Hill Climbing**
- **Stochastic Search**
- **Simulated Annealing**
- **Genetic Algorithms**

- **Class of algorithms applicable to many practical problems**
- **Not useful if more direct search methods can be used**
- **The algorithms are general black-boxes. What makes them work is the correct engineering of the problem representation**
  - **State representation**
  - **Neighborhoods**
  - **Evaluation function**
  - **Additional knowledge and heuristics**

(Some) References

- Russell & Norvig, Chap. 4
- Aarts & Lenstra. Local Search in Combinatorial Optimization. Wiley-InterScience. 1997.
- Spall. Introduction to Stochastic Search and Optimization. Wiley-InterScience. 2003.
- Numerical Recipes (http://www.nr.com/).
- Haupt&Haupt. Practical Genetic Algorithms. Wiley-InterScience. 2004.
- Mitchell. An Introduction to Genetic Algorithms (Complex Adaptive Systems). MIT Press. 2003.
- http://www.cs.washington.edu/homes/kautz/walksat/