Note: **Do not distribute slides or materials!**

- All slides and images retain copyright by their creators and are used here for educational purposes.

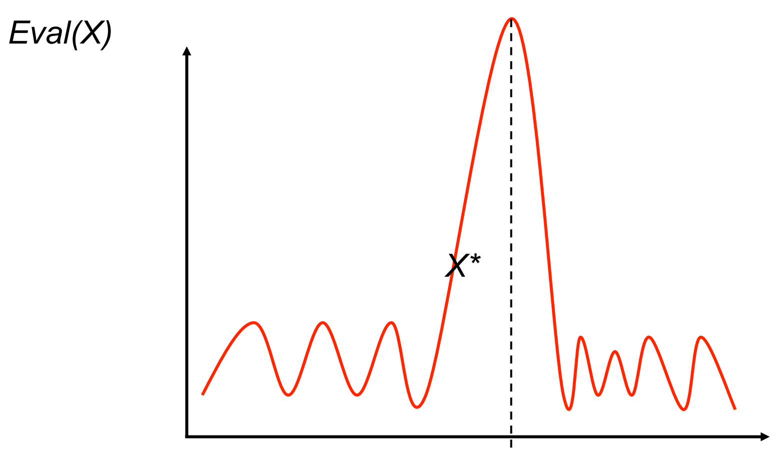
ARTIFICIAL INTELLIGENCE
Administration

- **Quiz results: very informative!**
- **Reading: Expected to read in advance of lecture**
- **Homework**: Can use 8 late days, at most 2 per homework
- **No laptops or cell phones**
  - **We expect total participation in class**
- **Extra credit:** Participation through
  - Clickers/question response
  - in-class question/answer
  - Will ask students randomly by name– please help me with pronounciation!!!

Optimization: Local Search & Stochastic Search

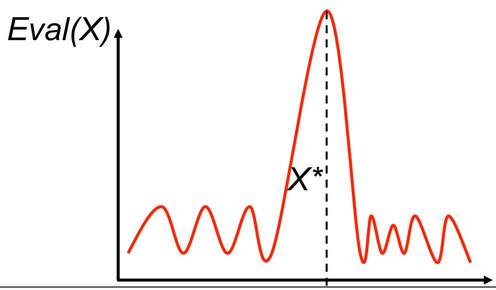
Drew Bagnell

(With thanks to Andrew Moore, Martial Hebert, Illah Nourbakhsh, Alexandre Bayen, Matt Zucker, and many more for slides...)

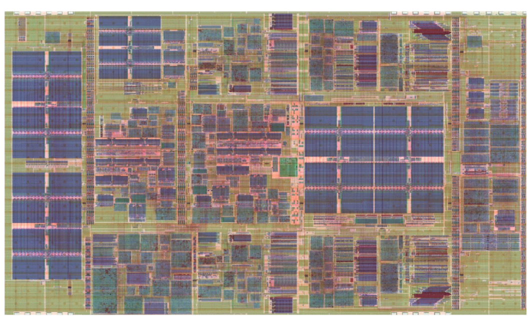


Today's Class of Search Problems

- Given:
  - A set of states (or configurations) $S = \{X_1..X_M\}$
  - A function that evaluates each configuration: *Eval*(X)
- Solve:
  - Find global extremum: Find X^* such that *Eval*(X^*) is greater than all *Eval*(X_i) for all possible values of X_i



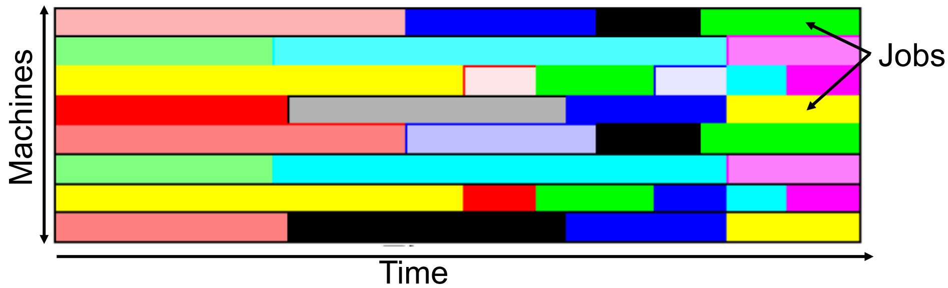
Real-World Examples



Placement
Floorplanning
Channel routing
Compaction

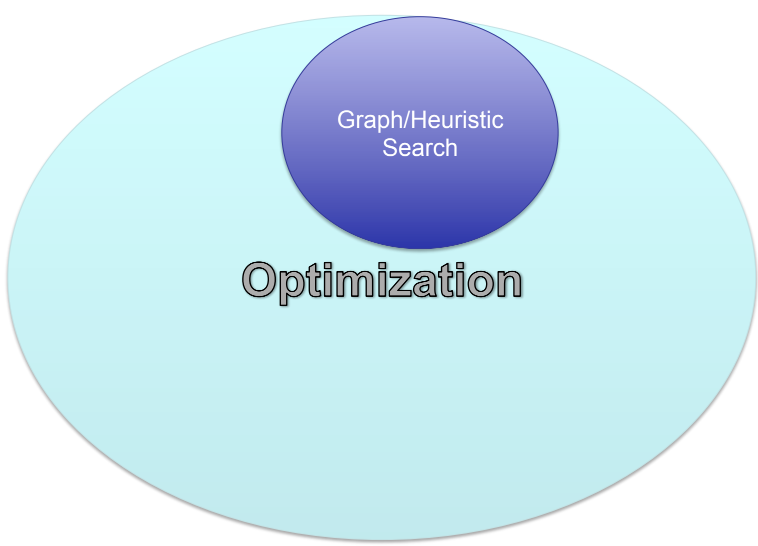
- VLSI layout:
  - X = placement of components + routing of interconnections
  - *Eval* = Distance between components + % unused + routing length

Real-World Examples

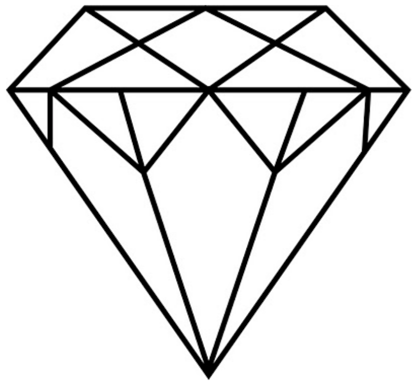


- Scheduling: Given *m* machines, *n* jobs
- *X* = assignment of jobs to machines
- *Eval* = completion time of the *n* jobs (minimize)

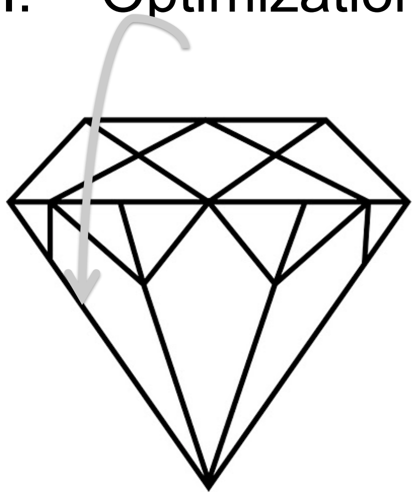
- Others: Vehicle routing, design, treatment sequencing, ………



A.I. = Optimization



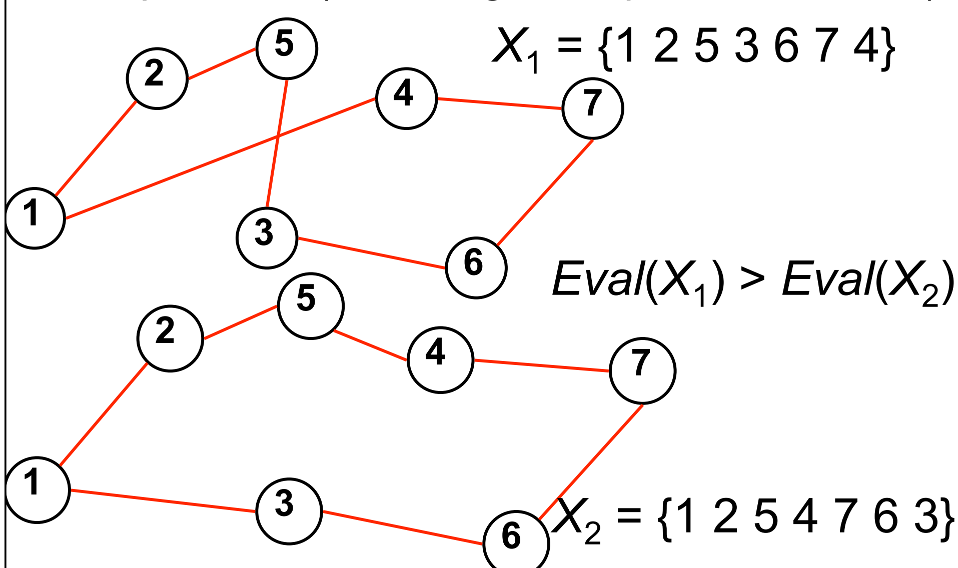
A.I. = Optimization



What makes this challenging?

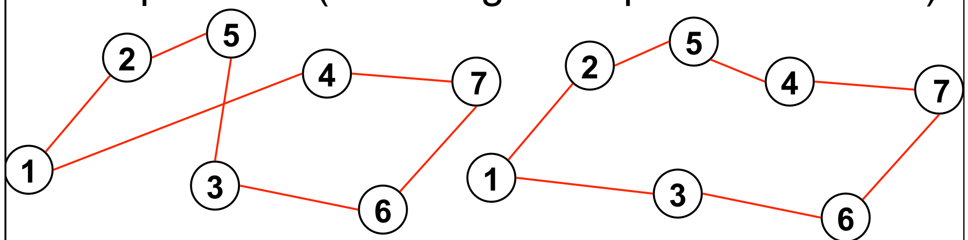
- Problems of particular interest:
  - Set of configurations too large to be enumerated explicitly
  - Computation of *Eval*(.) may be expensive
  - There is no (known) algorithm for finding the maximum of *Eval*(.) efficiently
  - Solutions with similar values of *Eval*(.) are considered equivalent for the problem at hand
  - *We do not care how we get to X*, we care only about the description of the configuration X**

Example: TSP (Traveling Salesperson Problem)



- Find a tour of minimum length passing through each point once

Example: TSP (Traveling Salesperson Problem)



X_1 = {1 2 5 3 6 7 4} X_2 = {1 2 5 4 7 6 3}

$Eval(X_1) > Eval(X_2)$

- Configuration X = tour through nodes {1,..,N}
- *Eval* = Length of path defined by a permutation of {1,..,N}
- Find X^* that realizes the *minimum* of $Eval(X)$
- Size of search space = order $(N\text{-}1)!/2$
- Note: Solutions for N = hundreds of thousands

Example: SAT (SATisfiability)

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

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

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

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

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

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

Example: SAT (SATisfiability)

A ∨ ¬**B** ∨ **C**

¬**A** ∨ **C** ∨ **D**

B ∨ **D** ∨ ¬**E**

¬**C** ∨ ¬**D** ∨ ¬**E**

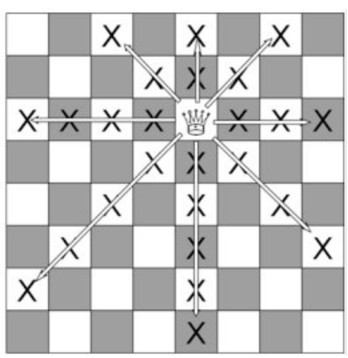
¬**A** ∨ ¬**C** ∨ **E**

....

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

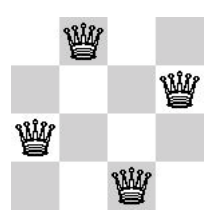
- Configuration X = Vector of assignments of N Boolean variables
- *Eval*(X) = Number of clauses that are satisfied given the assignments in X
- Find X^* that realizes the *maximum* of *Eval*(X)
- Size of search space = 2^N
- Note: Solutions for 1000s of variables and clauses

N-queens

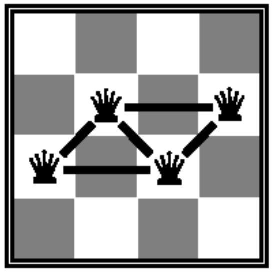


Valid queen moves

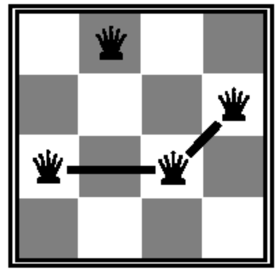
Valid 4-queen configuration



Example: N-Queens

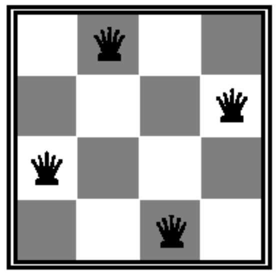


Eval(X) = 5



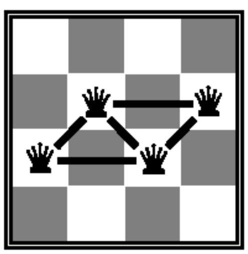
Eval(X) = 2

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

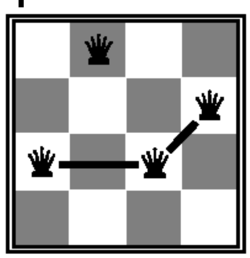


Eval(X) = 0

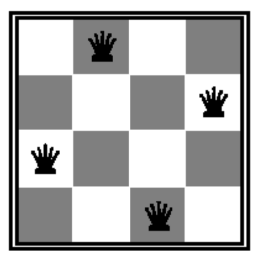
Example: N-Queens



Eval(X) = 5



Eval(X) = 2



Eval(X) = 0

- Configuration X = Position of the N queens in N columns
- *Eval*(X) = Number of pairs of queens that are attacking each other
- Find X^* that realizes the minimum: *Eval*(X^*) = 0
- Size of search space: order N^N
- Note: Solutions for N = millions

Local Search

- Assume that for each configuration *X*, we define a neighborhood (or "*moveset*") *Neighbors*(*X*) that contains the set of configurations that can be reached from *X* in one "move".

X_{o} , ← Initial state

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

Local Search

The definition of the neighborhoods is not obvious or unique in general. The performance of the search algorithm depends critically on the definition of the neihborhood which is not straightforward in general.

ial state

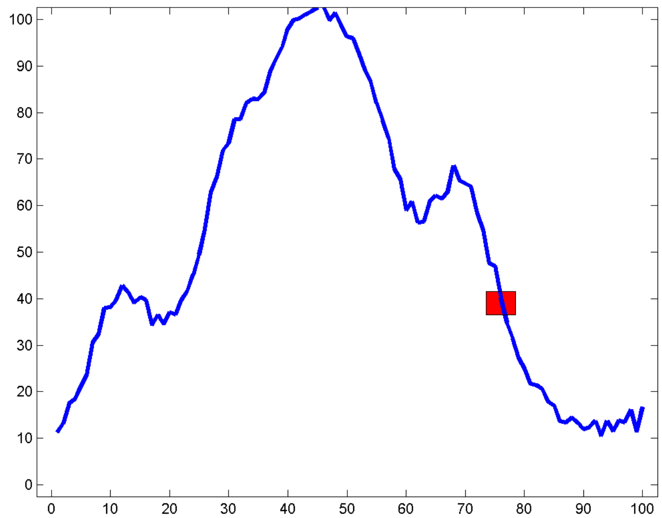
ntil we are "satisfied" with the
onfiguration:

3. Evalu te some of the ne hbors in *Neighbors*(X_i)
4. Select one of the neighbo X_{i+1}
5. Move to

Ingredient 1. Selection strategy: How to decide which neighbor to accept

Ingredient 2. Stopping condition

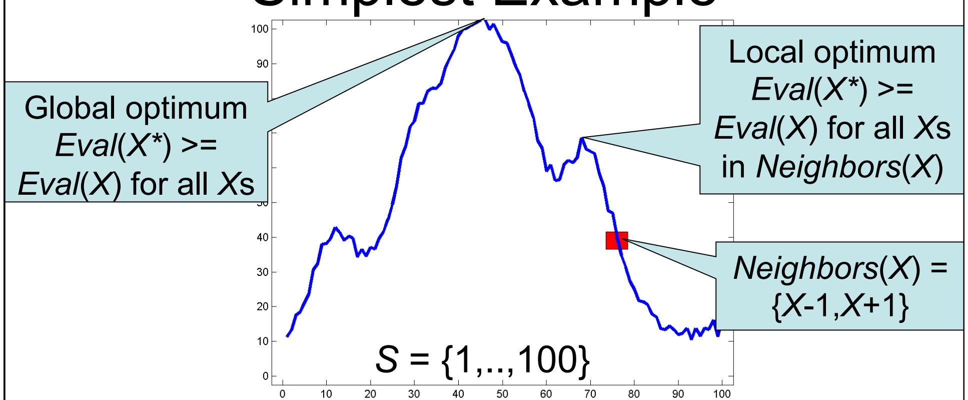
Simplest Example



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

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

Simplest Example



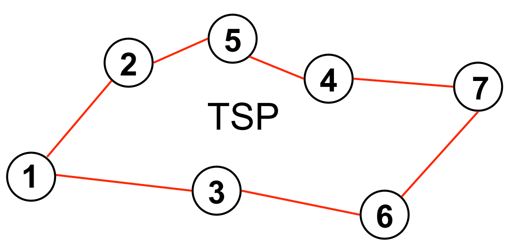
- We are interested in the *global* maximum, but we may have to be satisfied with a *local* maximum
- In fact, at each iteration, we can check only for *local* optimality
- The challenge: Try to achieve global optimality through a sequence of local moves

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

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

More Interesting Examples

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



TSP

SAT

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

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

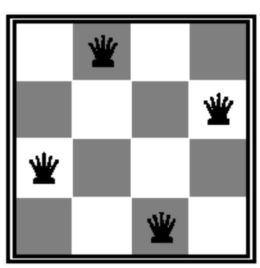
$\mathbf{B} \vee \mathbf{D} \vee \neg \mathbf{E}$

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

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

………

N-Queens

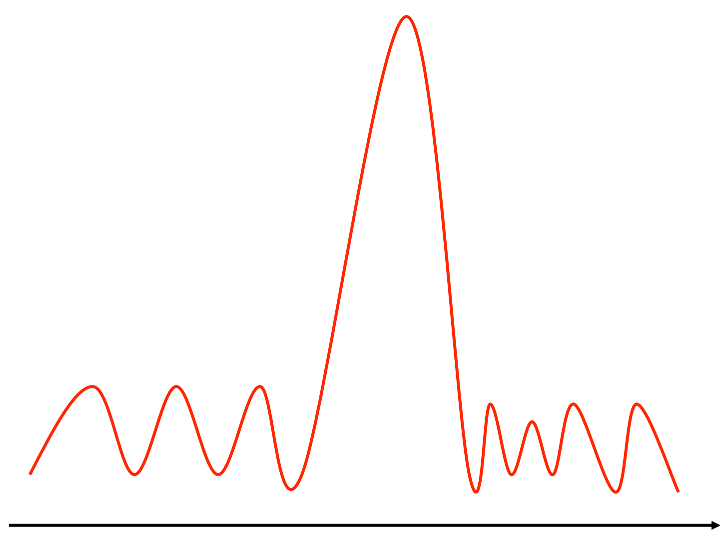


Issues

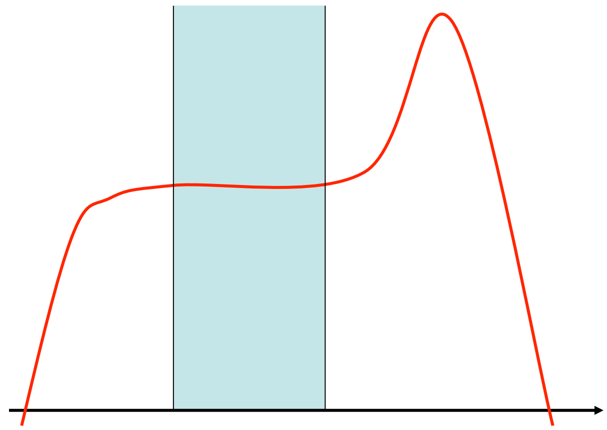
- Trade-off on size of neighborhood

→ larger neighborhood = better chance of finding a good maximum but may require evaluating an enormous number of moves

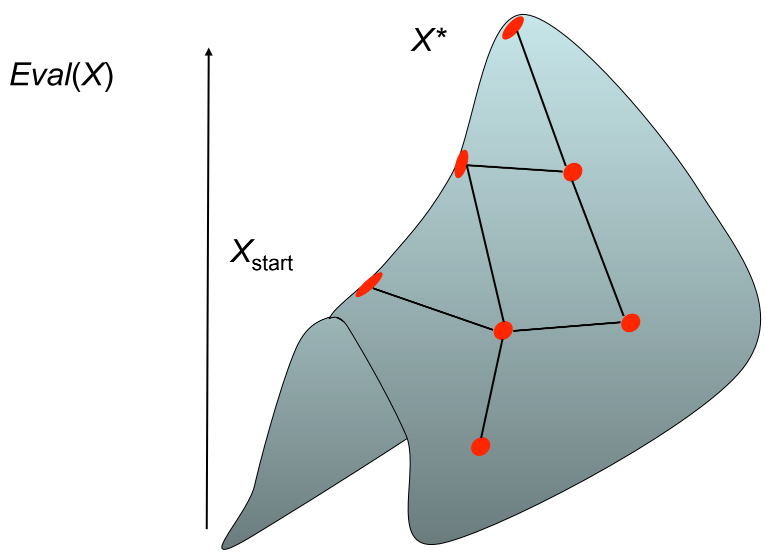
→ smaller neighborhood = smaller number of evaluations but may get stuck in poor local maxima



Multiple "poor" local maxima



Plateau = constant region of *Eval*(.)



Issues

- Constant memory usage
- All we can hope is to find the local maximum "closest" to the initial configuration → Can we do better than that?
- Ridges and plateaux will plague all local search algorithms

Issues

- Constant memory usage
- All we can hope is to find the local maximum "closest" to the initial configuration → Can we do better than that?
- Ridges and plateaux will plague all local search algorithms
- Design of neighborhood is critical (as important as design of search algorithm)
- 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 evaluation but may get stuck in poor local maxima

Stochastic Search: Randomized Hill-Climbing

- $X \leftarrow$ 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 $E' > E$

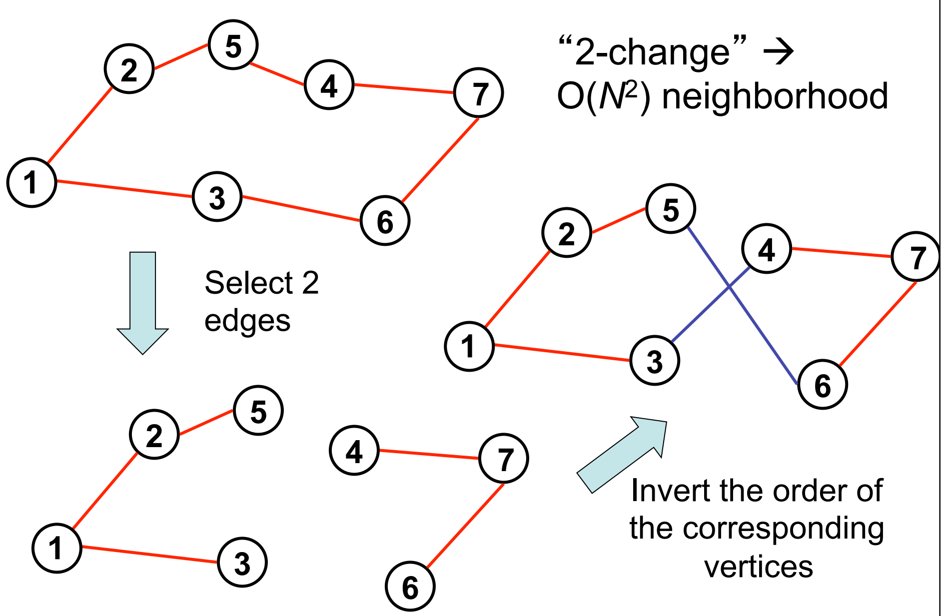
   $X \leftarrow X'$

   $E \leftarrow E'$

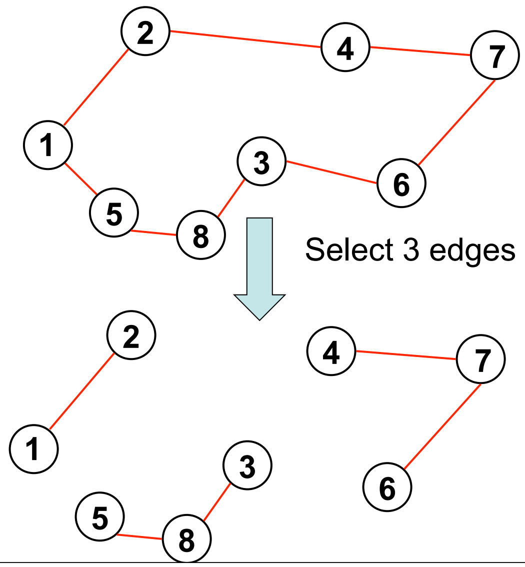
Critical change: We no longer select the best move in the entire neighborhood

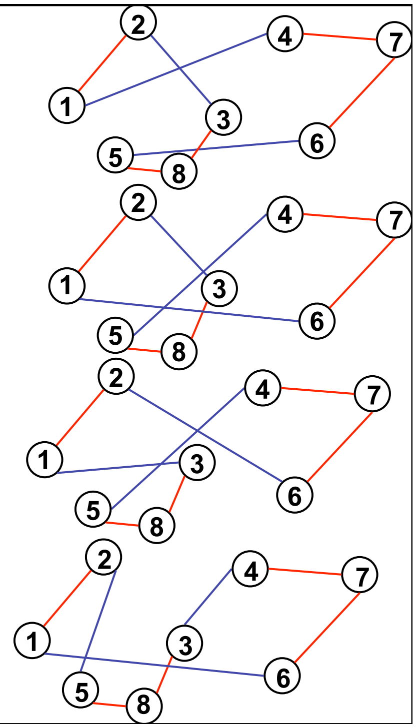
TSP Moves

"2-change" → O(N^2) neighborhood



“3-change” → O(N^3) neighborhood
…….. *k*-change





Hill-Climbing: TSP Example

	% error from min cost (N=100)	% error from min cost (N=1000)	Running time (N=100)	Running time (N=1000)
2-Opt	4.5%	4.9%	1	11
2-Opt (Best of 1000)	1.9%	3.6%		
3-Opt	2.5%	3.1%	1.2	13.7
3-Opt (Best of 1000)	1.0%	2.1%		

Data from: Aarts & Lenstra, “Local Search in Combinatorial Optimization”, Wiley Interscience Publisher

Hill-Climbing: TSP Example

- k-opt = Hill-climbing with k-change neighborhood
- Some results:
  - 3-opt better than 2-opt
  - 4-opt not substantially better given increase in computation time
  - Use random restart to increase probability of success
  - Better measure: % away from (estimated) minimum cost

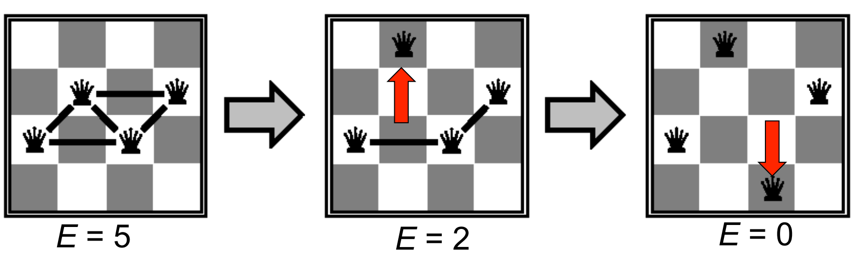
	% error from min cost (N=100)	% error from min cost (N=1000)	Running time (N=100)	Running time (N=1000)
2-Opt	4.5%	4.9%	1	11
2-Opt (Best of 1000)	1.9%	3.6%		
3-Opt	2.5%	3.1%	1.2	13.7
3-Opt (Best of 1000)	1.0%	2.1%		

Data from: Aarts & Lenstra, "Local Search in Combinatorial Optimization", Wiley Interscience Publisher

Hill-Climbing: N-Queens

- Basic hill-climbing is not very effective
- Exhibits plateau problem because many configurations have the same cost
- Multiple random restarts is standard solution to boost performance

N = 8	% Success	Average number of moves
Direct hill climbing	14%	4
With sideways moves	94%	21 (success)/64 (failure)



Data from Russell & Norvig

Hill-Climbing: SAT

$\mathbf{A} \vee \neg \mathbf{B} \vee \mathbf{C}$ $\quad$ $\neg \mathbf{C} \vee \neg \mathbf{D} \vee \neg \mathbf{E}$

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

- State X = assignment of N boolean variables
- Initialize the variables ($x_1,..,x_N$) randomly to *true*/*false*

- Iterate until all clauses are satisfied or max iterations:
  1. Select an unsatisfied clause
  2. With probability *p:* — Random walk part
     Select a variable x_i at random
  3. With probability 1-*p:* — Greedy part
     Select the variable x_i such that changing it will unsatisfy the least number of clauses (Max of *Eval*(X))
  4. Change the assignment of the selected variable x_i

Hill-Climbing: SAT

- WALKSAT algorithm still one of the most effective for SAT
- Combines the two ingredients: random walk and greedy hill-climbing
- Incomplete search: Can never find out if the clauses are **not** satisfiable

For more details and useful examples/code: http://www.cs.washington.edu/homes/kautz/walksat/

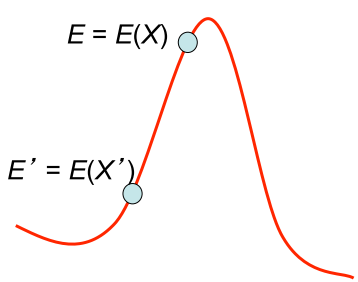
Optimization Method For Emergency Use Only

Simulated Annealing

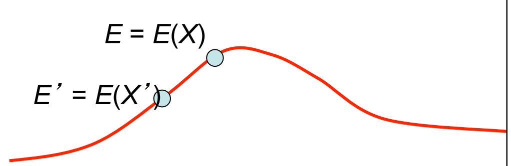
1. $E \leftarrow Eval(X)$
2. $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)
3. $E' \leftarrow Eval(X')$
4. If $E' >= E$
   $X \leftarrow X'$
   $E \leftarrow E'$
   Else accept the move to X' with some probability p:
   $X \leftarrow X'$
   $E \leftarrow E'$

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

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

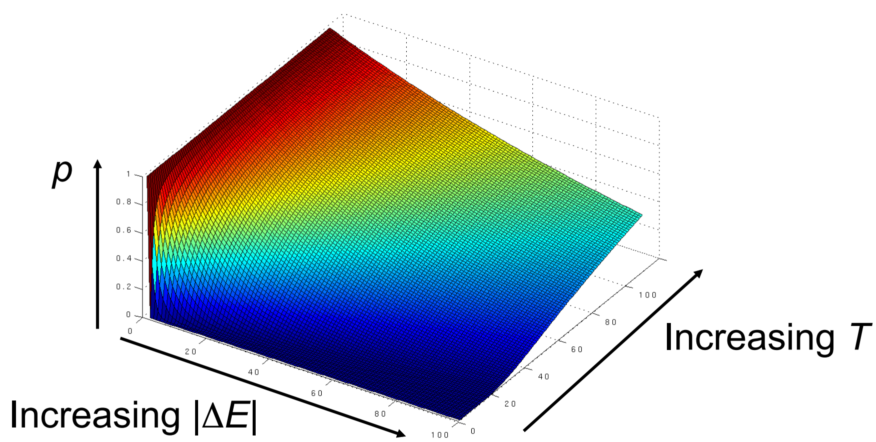
Choosing p: Simulated Annealing

- If $E' >= E$ accept the move

- Else accept the move with probability:

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

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



Simulated Annealing

1. Do K times:

    1.1 $E \leftarrow Eval(X)$

    1.2 $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)

    1.3 $E' \leftarrow Eval(X')$

    1.4 If $E' >= E$

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

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

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

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

Simulated Annealing

- $X \leftarrow$ Initial configuration
- $T \leftarrow$ Initial high temperature
- Iterate:

Iterate a number of times keeping the temperature fixed

1. Do K times:

    1.1 $E \leftarrow Eval(X)$

    1.2 $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)

    1.3 $E' \leftarrow Eval(X'$

Use the previous definition of the probability

1.4 If $E' >= E$

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

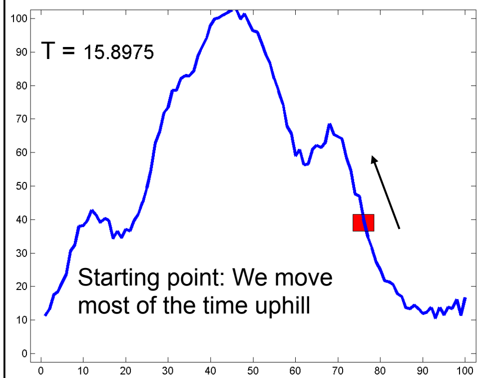
bility $p = e^{-(E - E')/T}$

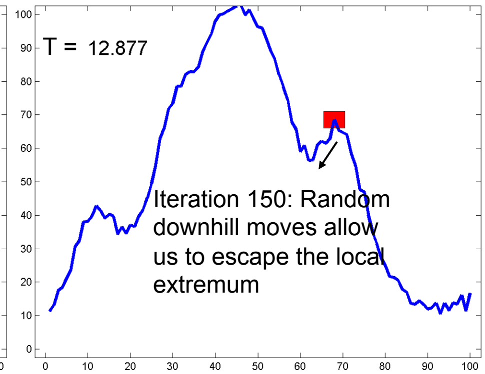
$X \leftarrow X'; E \leftarrow E';$

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

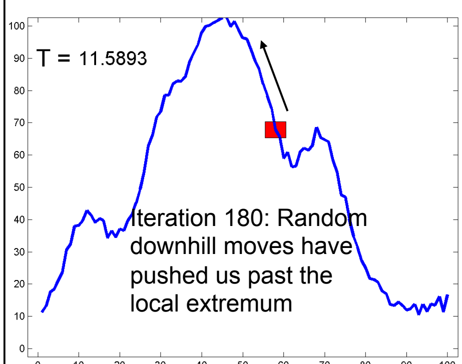
$T = 0 \rightarrow$ Greedy hill climbing
$T = \infty \rightarrow$ Random walk

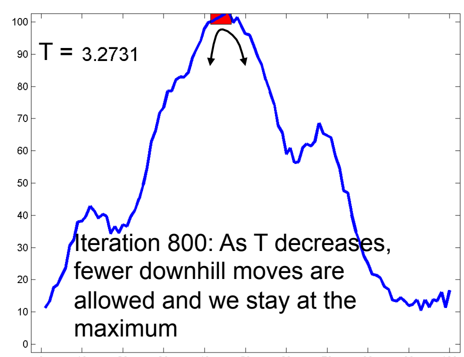
Basic Example



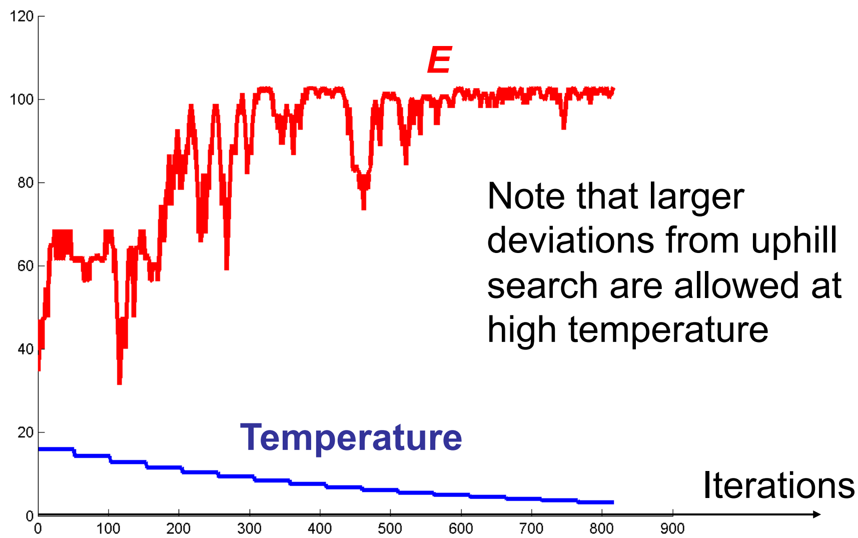


Basic Example





Basic Example



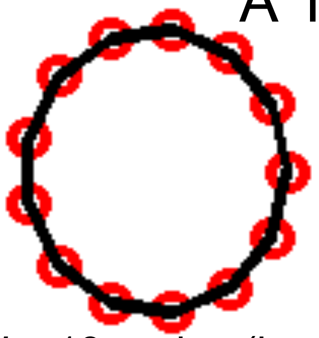
Where does this come from?

- 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(.)
- N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth. A.H. Teller and E. Teller, *Journal Chem. Phys.* **21** (1953) 1087-1092

A TSP Example



N = 13 nodes (in a circle)

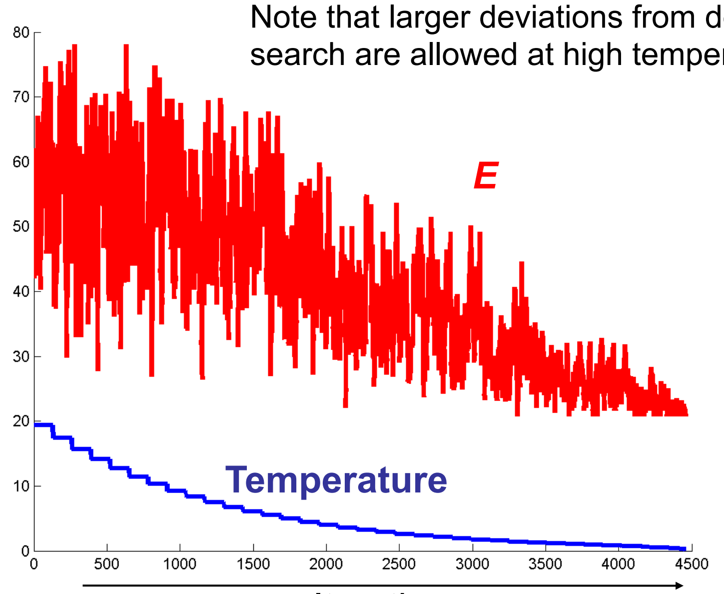
K = 100*N*
E = 25

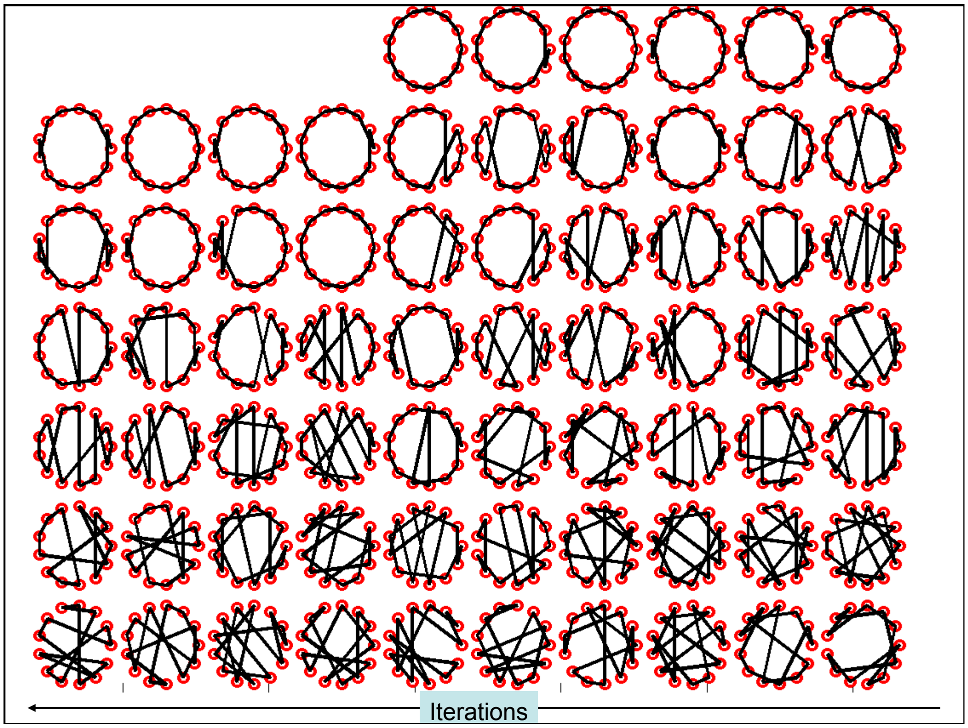
Starting configuration
E(*X*) = 55

Note: Boring but it has an obvious solution

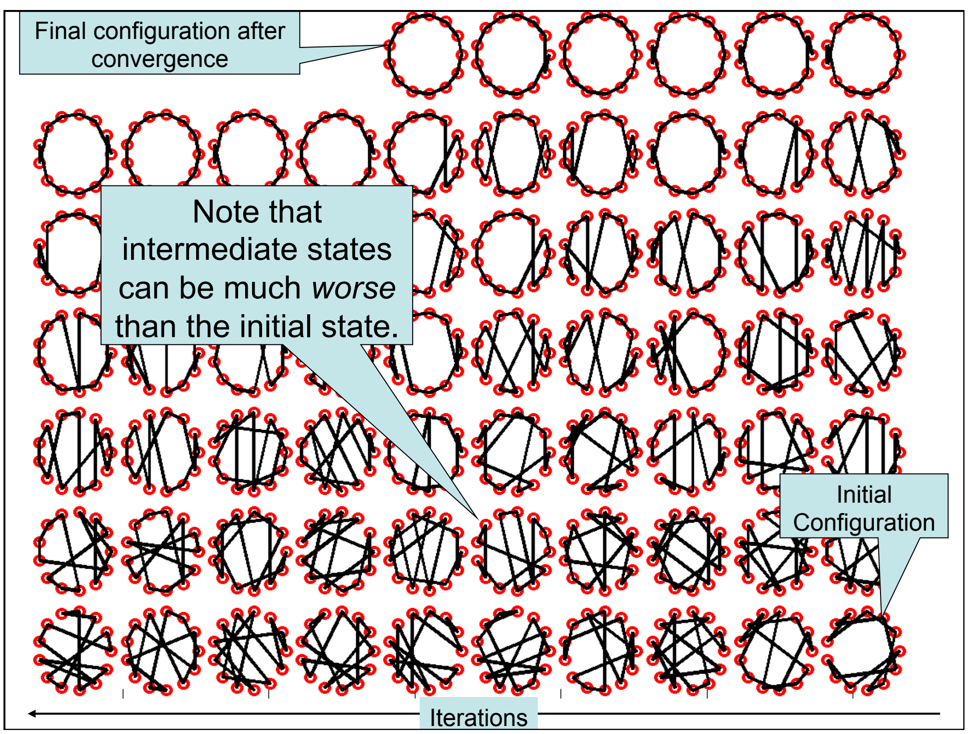
A TSP Example

Note that larger deviations from downhill search are allowed at high temperature



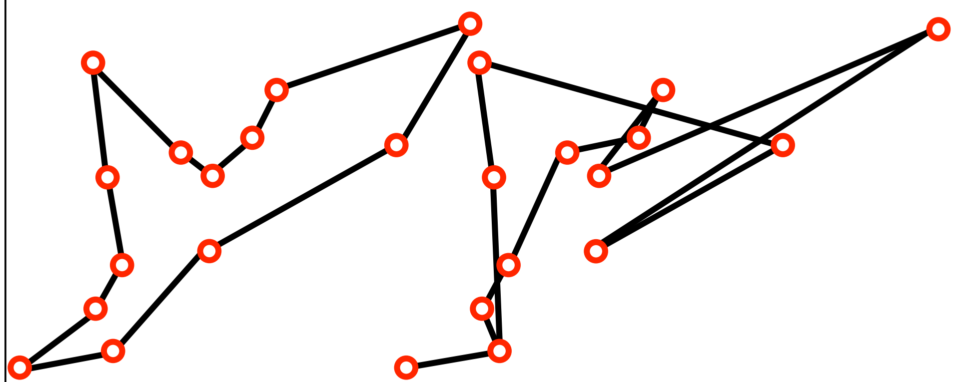


Iterations



Final configuration after convergence
Note that intermediate states can be much *worse* than the initial state.
Initial Configuration
Iterations

Another Example

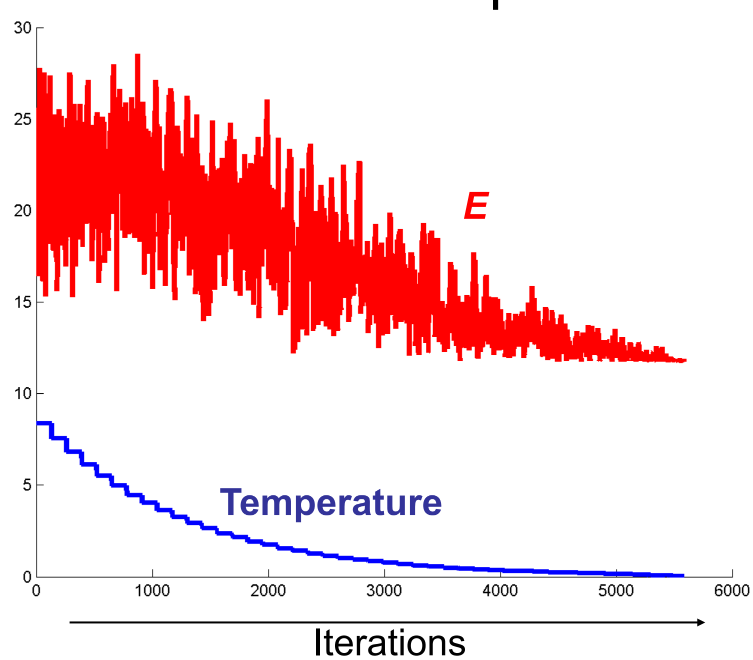


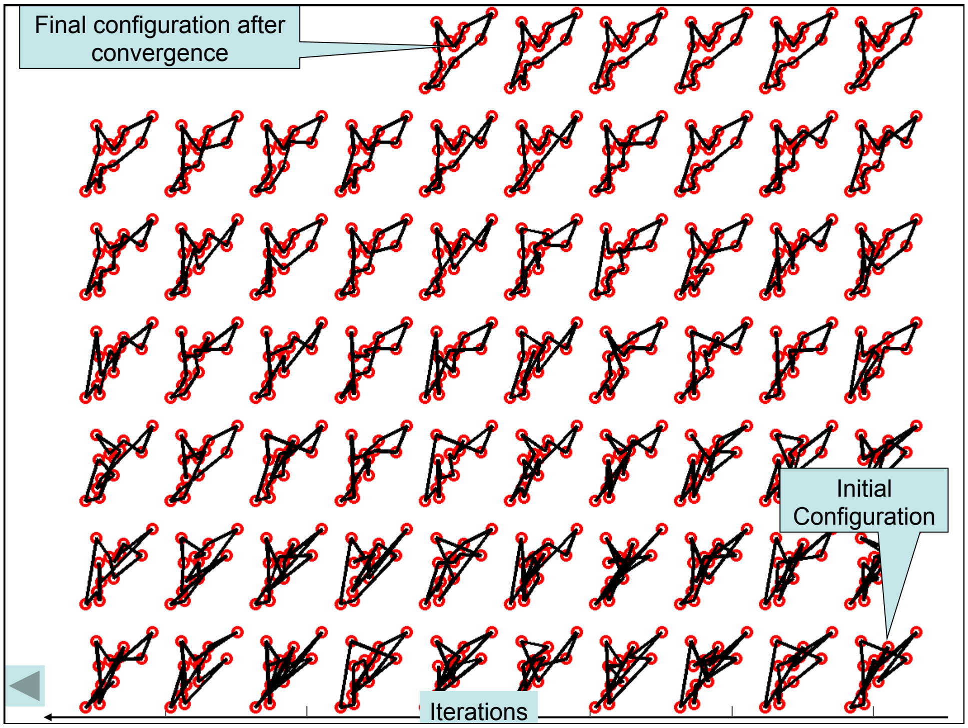
N = 13 nodes

Initial state

K = 100N

Another Example





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

Simulated Annealing

- $X \leftarrow$ Initial configuration
- $T \leftarrow$ Initial high temperature
- Iterate:

Many parameters need to be tweaked!!

1. Do K times:
   - 1.1 $E \leftarrow Eval(X)$
   - 1.2 $X' \leftarrow$ one configuration randomly selected in *Neighbors* (X)
   - 1.3 $E' \leftarrow Eval(X')$
   - 1.4 If $E' >= E$
     - $X \leftarrow X'; E \leftarrow E';$
     - Else accept the move with probability $p = e^{-(E - E')/T}$
     - $X \leftarrow X'; E \leftarrow E';$
2. $T \leftarrow \alpha T$

SA Discussion

- Design of neighborhood is critical
- *How to choose K?* Typically related to size of neighborhood
- *How to choose α?* Critical to avoid large number of useless evaluations. Especially a problem close to convergence (empirically, most of the time spent close to the optimum)

SA Discussion

- *How to choose starting temperature?* Typically related to the distribution of anticipated values of ΔE (e.g., T_{start} = max{ΔE over a large sample of pairs of neighbors})
- *What if we choose a really bad starting X?* Multiple random restart.
- *How to avoid repeated evaluation?* Use a bit more memory by remembering the previous moves that were tried ("Tabu search")
- Use (faster) approximate evaluation if possible (How?)

SA Discussion

- Often better than hill-climbing. Successful algorithm in many applications
- Many parameters to tweak. If not careful, may require very large number of evaluations
- Semi-infinite number of variations for improving performance depending on applications

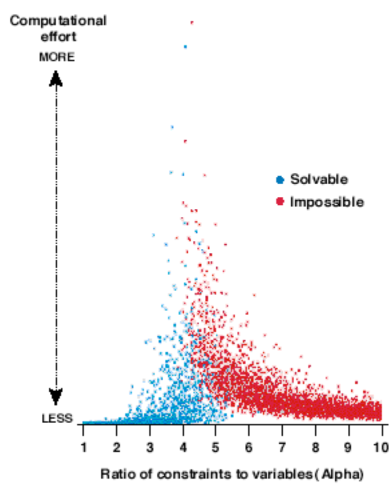
Phase Transitions–
The Physics of Optimization

- Over the last 20 years, physicists and computer scientists working in AI have discovered close connections between
  - the statistical behavior of matter
  - Computational hardness
- Consider the 3-SAT problem, and particularly the behavior in terms of
  - Alpha: ratio of constraints to variables
- What do you expect to happen?

http://www.nytimes.com/library/national/science/071399sci-satisfiability-problems.html

Phase Transitions–
The Physics of Optimization

Source: ***American Scientist***
Credit: The New York Times



Reminder Example: SAT (SATisfiability)

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

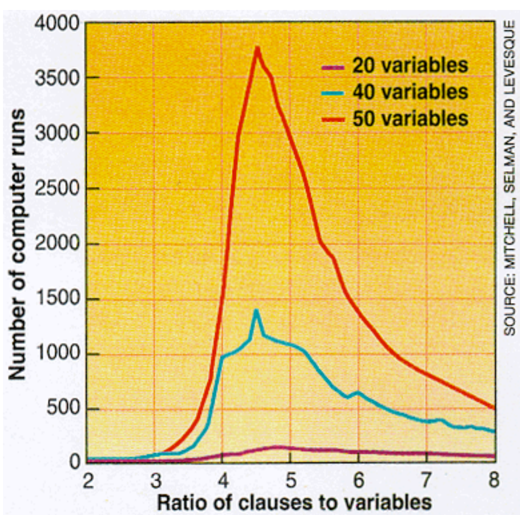
$\neg \mathbf{A} \vee \mathbf{C} \vee \mathbf{D}$

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

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

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

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



Mounting difficulties. The work of checking complex logical statements peaks at a threshold.

Genetic/Evolutionary Algorithms

Genetic Algorithms

- View optimization by analogy with evolutionary theory → Simulation of natural selection
- View configurations as *individuals* in a *population*
- View *Eval* as a measure of *fitness*
- Let the least-fit individuals die off without reproducing
- Allow individuals to *reproduce* with the best-fit ones selected more often
- Each *generation* should be overall better fit (higher value of Eval) than the previous one
- If we wait long enough the population should evolve so toward individuals with high fitness (i.e., maximum of *Eval*)

Genetic Algorithms: Implementation

- Configurations represented by strings:

$X =$ | 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

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 receive part of the genes from each of the parents
- Implemented by crossover operation

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

Offspring:

1 0 0 1 0 0 0 1

1 0 1 1 1 0 0 1

Genetic Algorithms: Mutation

1 0 0 1 1 0 0 1

1 0 0 1 0 0 0 1

1 1 1 1 1 0 0 1

Select a random individual

1 1 1 1 1 0 0 1

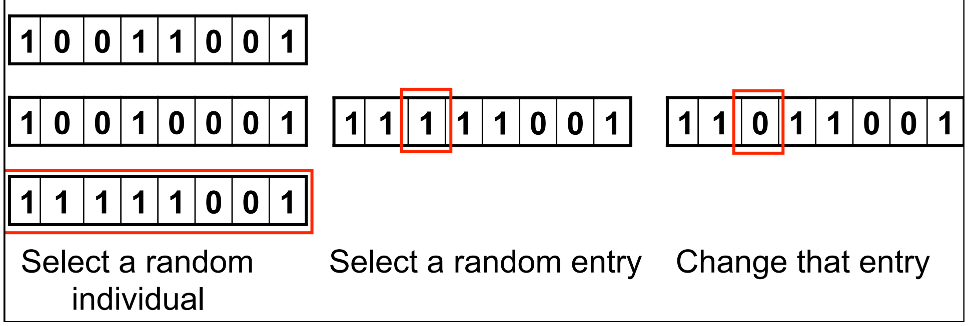
Select a random entry

1 1 0 1 1 0 0 1

Change that entry

Genetic Algorithms: Mutation

- Random change of one element in one configuration
  - →Implements random deviations from inherited traits
  - →Corresponds loosely to "random walk": Introduce random moves to avoid small 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 initial population $X = \{X_1,..,X_P\}$
- Iterate:

Stopping condition is not obvious?

1. Select K ra...
2. For each pair of parents ...

 1.1 Generate offsprings (Y_1...

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

Variation: Generate only one offspring

...r 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

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

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

- Example (*tournament*): Select a random small subset of the population and select the best-fit individual as a parent

- Implements “survival of the fittest”
- Corresponds loosely to the greedy part of hill–climbing (we try to move uphill)

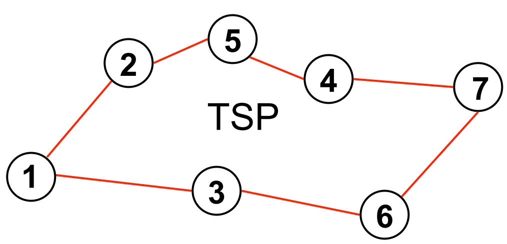
GA and Hill Climbing

- 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
     offspring Y_i:
     randomly selected element of the
     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 would you set up these problems to use GA search?



SAT

$A \vee \neg B \vee C$

$\neg A \vee C \vee D$

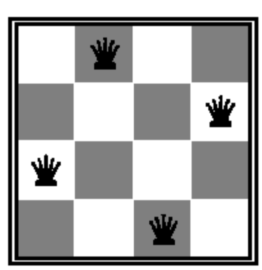
$B \vee D \vee \neg E$

$\neg C \vee \neg D \vee \neg E$

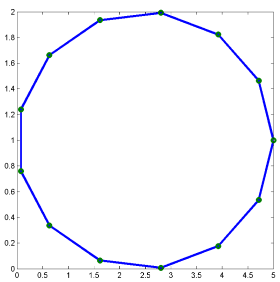
$\neg A \vee \neg C \vee E$

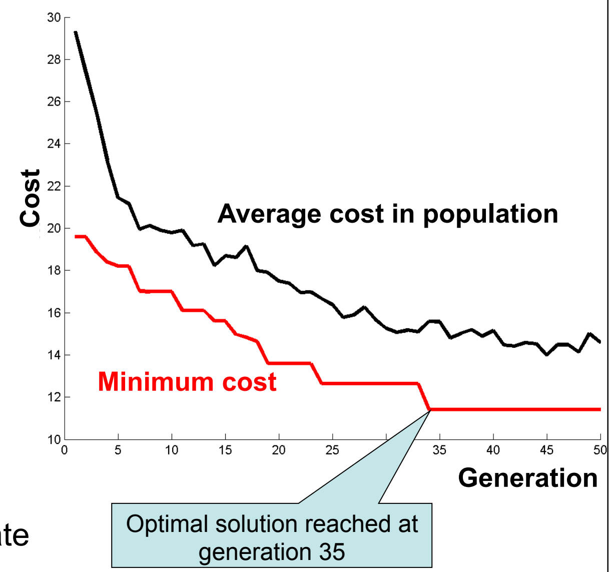
....

N-Queens



TSP Example

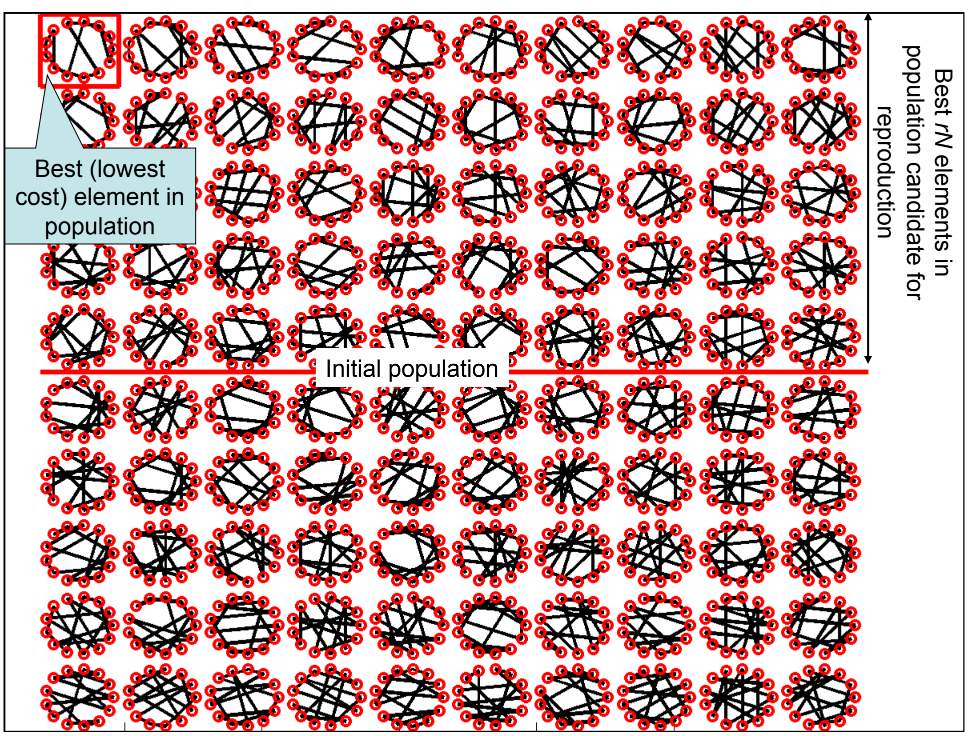


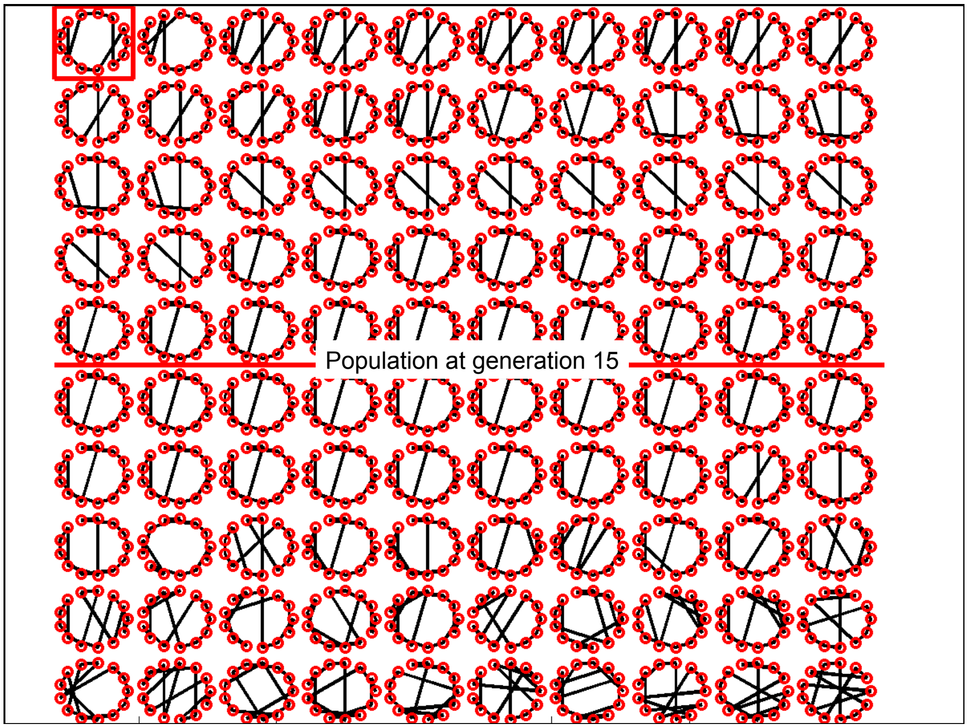


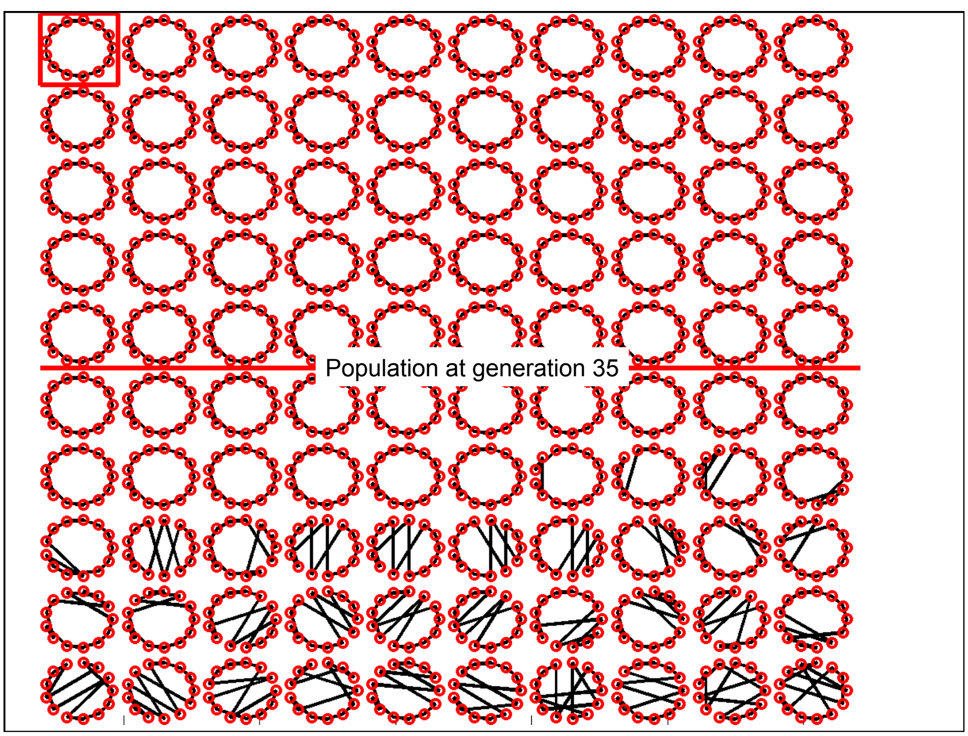
N = 13

P = 100 elements in population

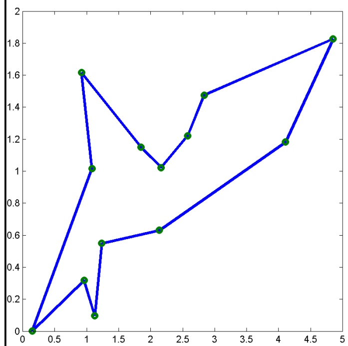
μ = 4% mutation rate
r = 50% reproduction rate




Population at generation 15


Population at generation 35

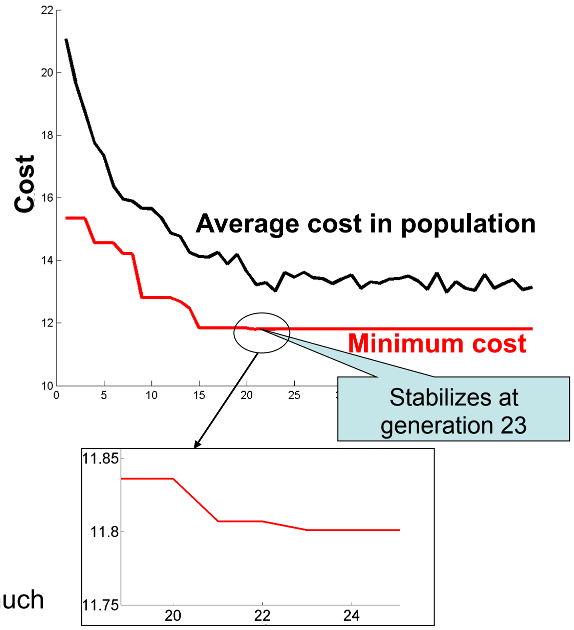
Another TSP Example

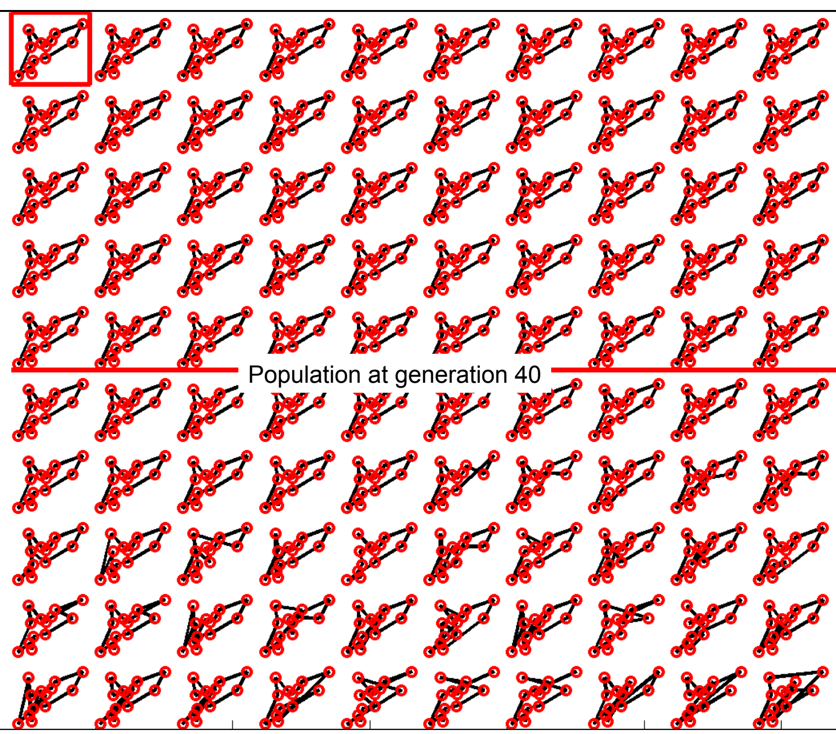


Converges and remains stable after generation 23

0.4% difference:
GA = 11.801
SA = 11.751

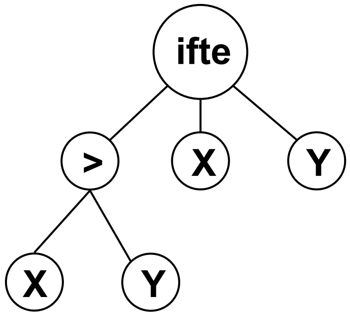
But: Number of operations (number of cost evaluations) much smaller (approx. 2500)



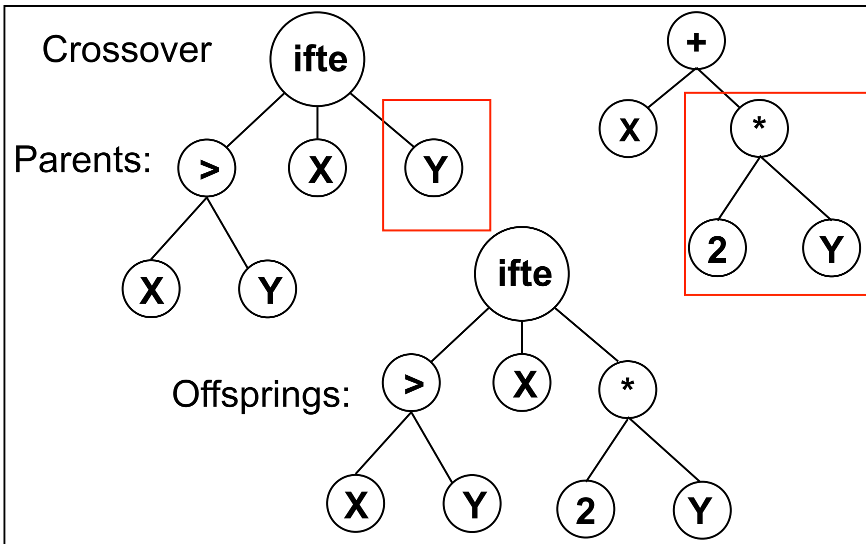


Even more radical ideas..

Individual = program
X = parse tree representing a program



(ifte (X > Y) X Y)



Use genetic algorithms as before with this definition of crossover
Example applications: robot controller, signal processing, circuit design
Intriguing, but alternative solutions exist for most of these applications; this is not the first approach to consider!!!

Koza. Genetic programming: On the programming of computers by means of natural selection. MIT Press. 1992
http://www.genetic-programming.org/

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

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.