# Hill Climbing and Simulated Annealing

**Jianlin Cheng, PhD**

**Computer Science Department**

**University of Missouri, Columbia**

**Fall, 2014**

# References of the Theory

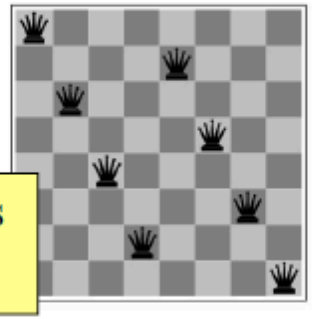
- Andrew Moore’s slides
- Xiaojin Zhu’s slides

# An Optimization Problem

- Each state s has a score *f*(s) (or cost, energy) that we can compute.
- The goal is to find the state with the highest score (or lowest cost, energy) – low / high should be obvious.
- Enumerating all the states is intractable (e.g. NP hard problem).

# Examples

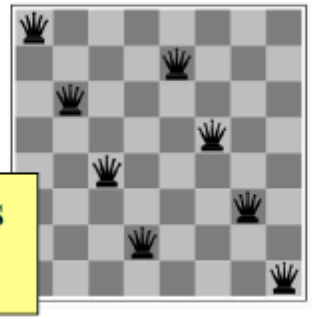
- N-queen: $f(s)$ = number of conflicting queens in state $s$



Note we want $s$ with the lowest score $f(s)=0$. The techniques are the same. Low or high should be obvious from context.

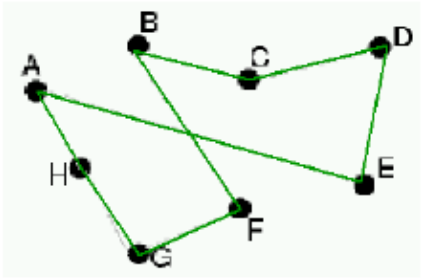
# Examples

- N-queen: $f(s)$ = number of conflicting queens in state $s$



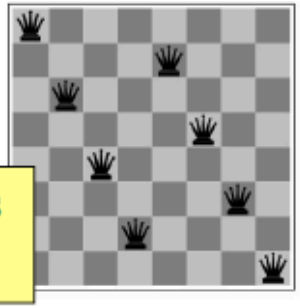
Note we want $s$ with the lowest score $f(s)=0$. The techniques are the same. Low or high should be obvious from context.

- Traveling salesperson problem (TSP)
  - Visit each city once, return to first city
  - State = order of cities, $f(s)$ = total mileage



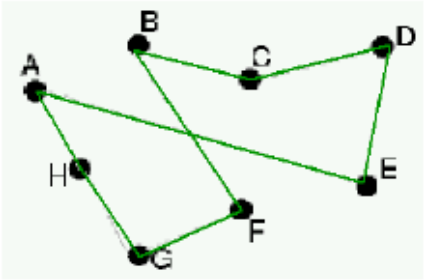
# Examples

- N-queen: $f(s)$ = number of conflicting queens in state $s$



Note we want $s$ with the lowest score $f(s)=0$. The techniques are the same. Low or high should be obvious from context.

- Traveling salesperson problem (TSP)
  - Visit each city once, return to first city
  - State = order of cities, $f(s)$ = total mileage




- Boolean satisfiability (e.g., 3-SAT)
  - State = assignment to variables
  - $f(s)$ = # satisfied clauses

$A \vee \neg B \vee C$
$\neg A \vee C \vee D$
$B \vee D \vee \neg E$
$\neg C \vee \neg D \vee \neg E$
$\neg A \vee \neg C \vee E$

# Iterative Optimization Methods

Start at a random configuration; repeatedly consider various moves; accept some & reject some. When you're stuck, restart.

We must invent a **moveset** that describes what moves we will consider from any configuration. Let's invent movesets for out four sample problems.

# 1. HILL CLIMBING

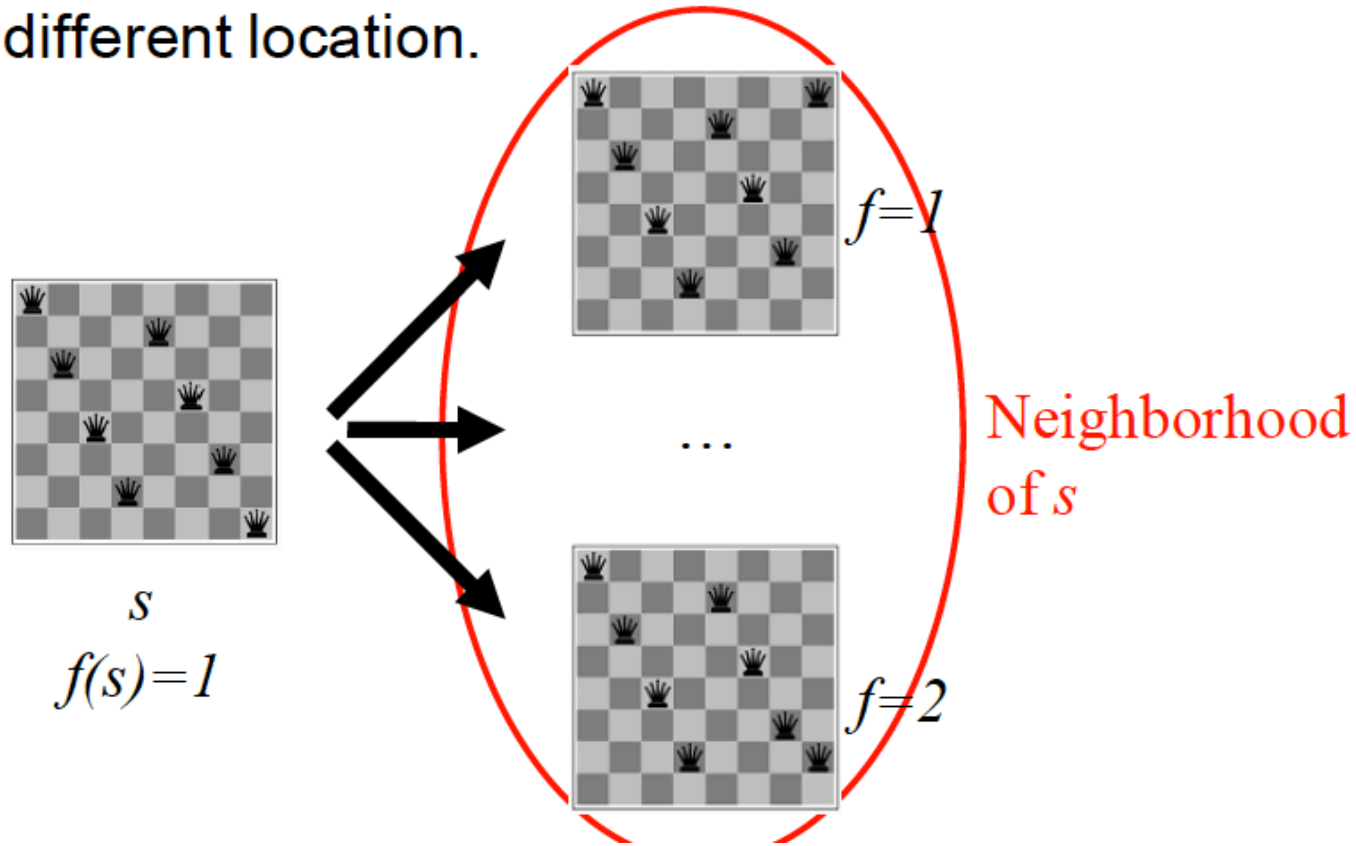
# Hill Climbing

- Very simple idea: Start from some state *s*,
  - Move to a neighbor *t* with better score. Repeat.
- **Question**: what's a neighbor?
  - You have to define that!
  - The neighborhood of a state is the set of neighbors
  - Also called 'move set'
  - Similar to successor function

# Neighbors: N-Queen

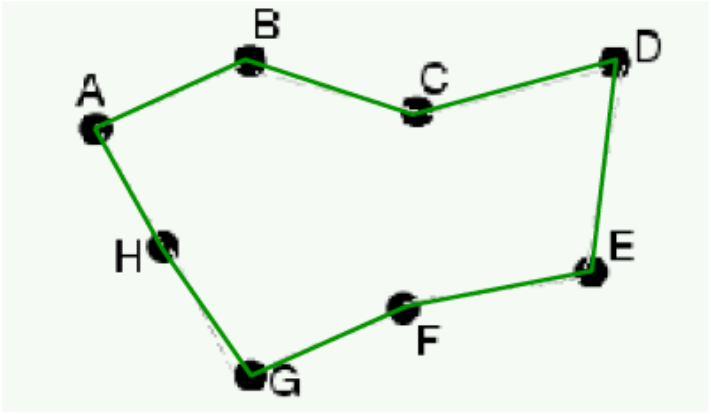
Example: N-queen (one queen per column). One possibility: tie breaking more promising?

- Pick the right-most most-conflicting column;
- Move the queen in that column vertically to a different location.



# Neighbors: TSP

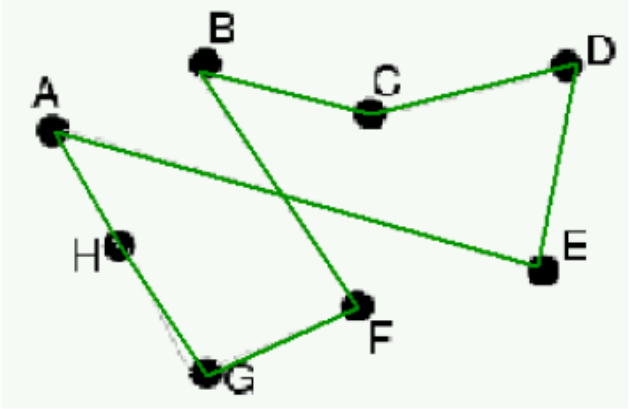
- state: A-B-C-D-E-F-G-H-A
- *f* = length of tour
- One possibility: 2-change




A-B-C-D-E-F-G-H-A

flip

A-E-D-C-B-F-G-H-A



# Neighbors: SAT

- State: (A=T, B=F, C=T, D=T, E=T)
- *f* = number of satisfied clauses
- Neighbor:

$A \vee \neg B \vee C$
$\neg A \vee C \vee D$
$B \vee D \vee \neg E$
$\neg C \vee \neg D \vee \neg E$
$\neg A \vee \neg C \vee E$

# Neighbors: SAT

- State: (A=T, B=F, C=T, D=T, E=T)
- *f* = number of satisfied clauses
- Neighbor: flip the assignment of one variable

(A=**F**, B=F, C=T, D=T, E=T)
(A=T, B=**T**, C=T, D=T, E=T)
(A=T, B=F, C=**F**, D=T, E=T)
(A=T, B=F, C=T, D=**F**, E=T)
(A=T, B=F, C=T, D=T, E=**F**)

$A \vee \neg B \vee C$
$\neg A \vee C \vee D$
$B \vee D \vee \neg E$
$\neg C \vee \neg D \vee \neg E$
$\neg A \vee \neg C \vee E$

# Hill Climbing

- **Question**: What's a neighbor?
  - (vaguely) Problems tend to have structures. A small change produces a neighboring state.
  - The neighborhood must be small enough for efficiency
  - Designing the neighborhood is critical. This is the real ingenuity – not the decision to use hill climbing.
- **Question**: Pick which neighbor?
- **Question**: What if no neighbor is better than the current state?

# Hill Climbing

- **Question**: What's a neighbor?
  - (vaguely) Problems tend to have structures. A small change produces a neighboring state.
  - The neighborhood must be small enough for efficiency
  - Designing the neighborhood is critical. This is the real ingenuity – not the decision to use hill climbing.
- **Question**: Pick which neighbor? The best one (greedy)
- **Question**: What if no neighbor is better than the current state? Stop. (Doh!)

# Hill Climbing Algorithm

Hill-climbing: Attempt to maximize Eval(X) by moving to the highest configuration in our moveset. If they're all lower, we are stuck at a "local optimum."

1. Let X := initial config
2. Let E := Eval(X)
3. Let N = moveset_size(X)
4. For ( i = 0 ; i<N ; i := i+1)
   Let $E_i$ := Eval(move(X,i))
5. If all $E_i$'s are ≤ E, terminate, return X
6. Else let i* = $\text{argmax}_i$ $E_i$
7. X := move(X,i*)
8. E := $E_{i^*}$
9. Goto 3

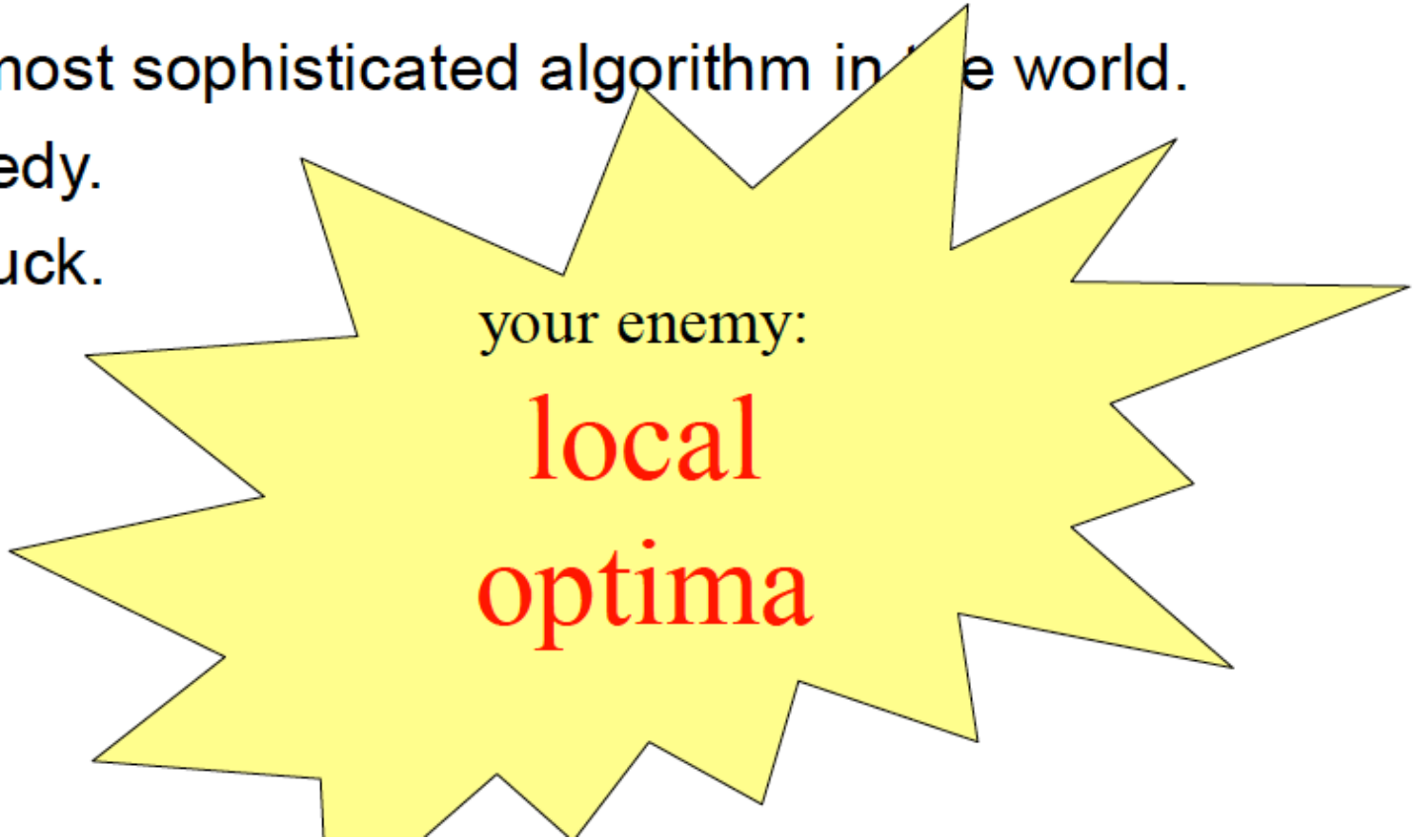
(Not the most sophisticated algorithm in the world.)

# Hill Climbing Issues

- Trivial to program
- Requires no memory (since no backtracking)
- MoveSet design is critical. This is the real ingenuity – not the decision to use hill-climbing.
- Evaluation function design often critical.
  - Problems: dense local optima or plateaux
- If the number of moves is enormous, the algorithm may be inefficient. What to do?
- If the number of moves is tiny, the algorithm can get stuck easily. What to do?
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Does hill-climbing permit that?
- What if approximate evaluation is cheaper than accurate evaluation?
- Inner-loop optimization often possible.

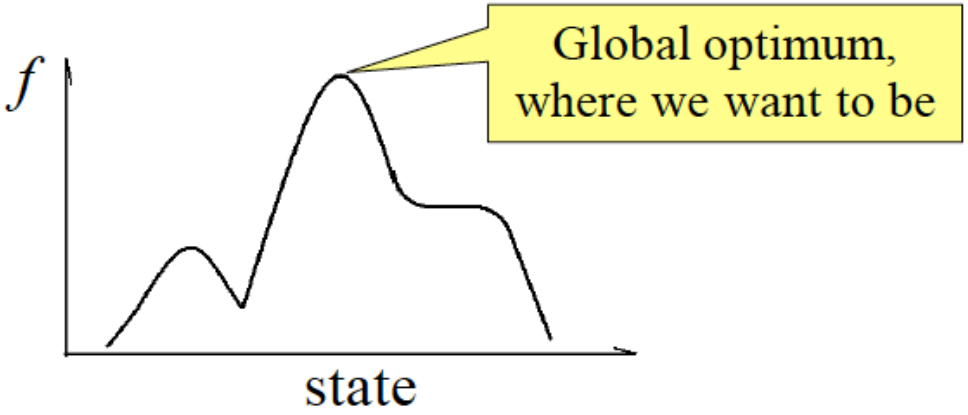
# Major Problems

- Not the most sophisticated algorithm in the world.
- Very greedy.
- Easily stuck.

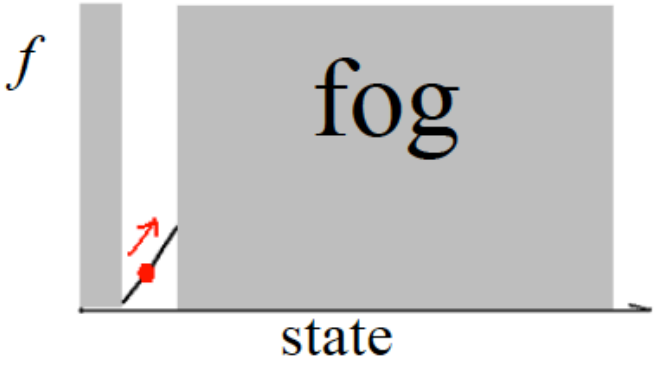


# Local Optima in Hill Climbing

- Useful conceptual picture: *f* surface = 'hills' in state space

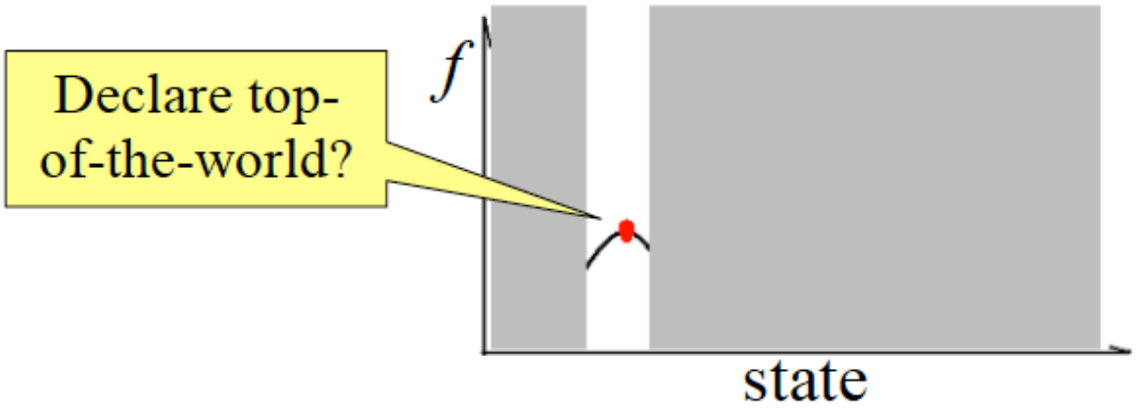



- But we can't see the landscape all at once. Only see the neighborhood. Climb in fog.

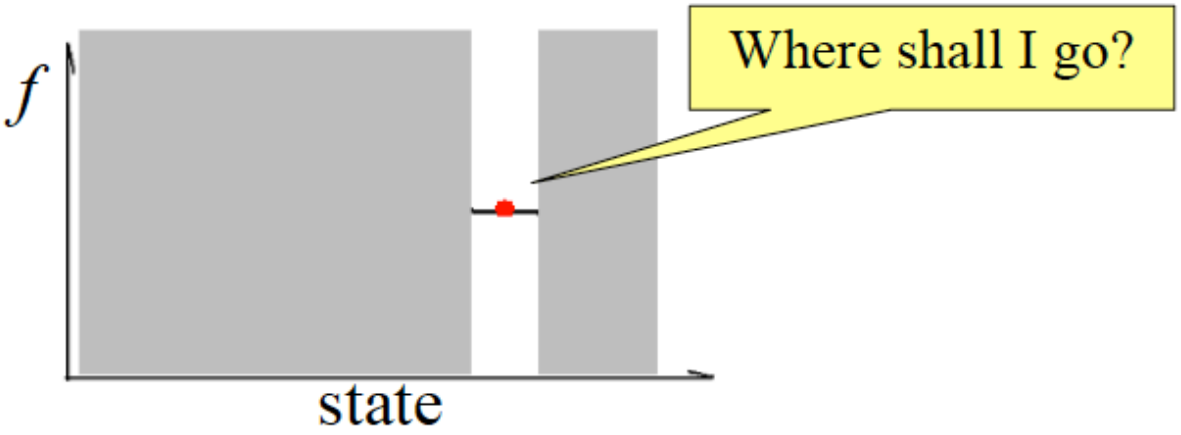


# Local Optima in Hill Climbing

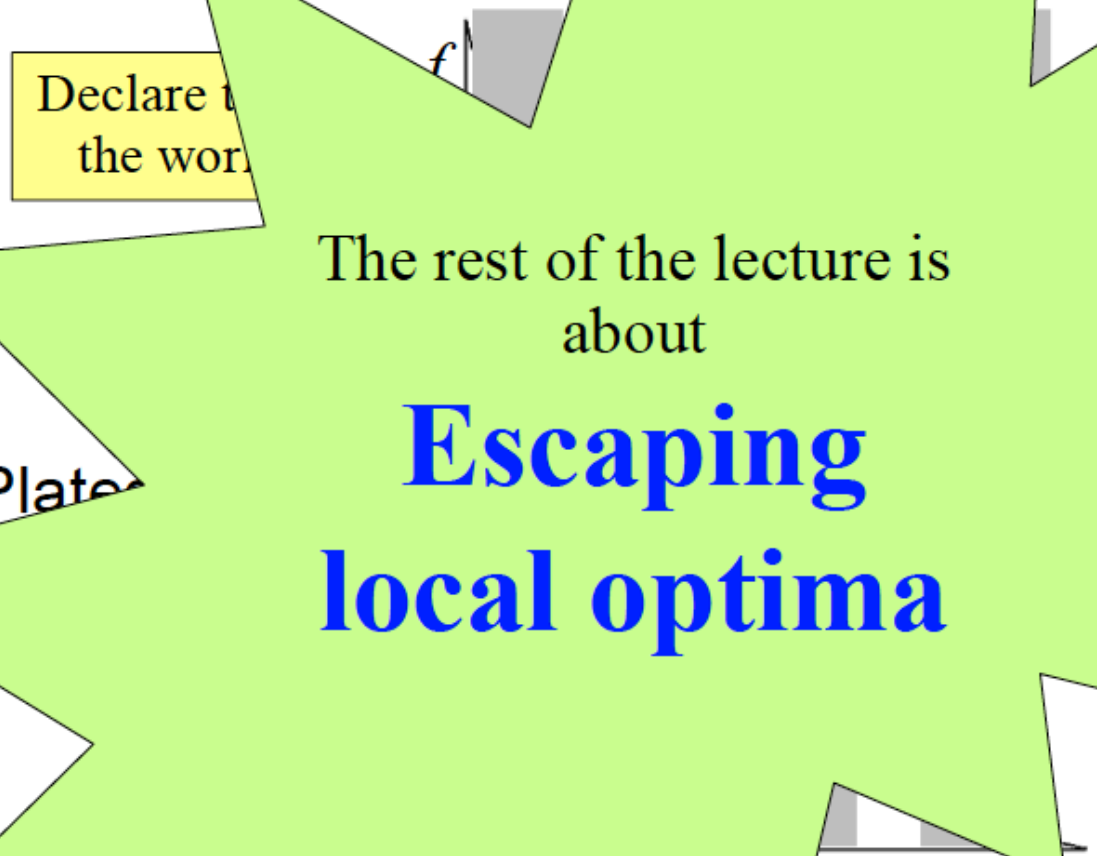
- Local optima (there can be many!)




- Plateaux



- Local optima (there ca e ma




- Plate

The rest of the lecture is about

# Escaping local optima

# Randomized Hill Climbing

1. Let X := initial config
2. Let E := Eval(X)
3. Let i = random move from the moveset
4. Let $E_i$ := Eval(move(X,i))
5. If E < $E_i$ then
   X := move(X,i)
   E := $E_i$
6. Goto 3 unless bored.

What stopping criterion should we use?

Any obvious pros or cons compared with our previous hill climber?

# Repeat Hill Climbing with Random Start

- Very simple modification

1. When stuck, pick a random new start, run basic hill climbing from there.
2. Repeat this *k* times.
3. Return the best of the *k* local optima.

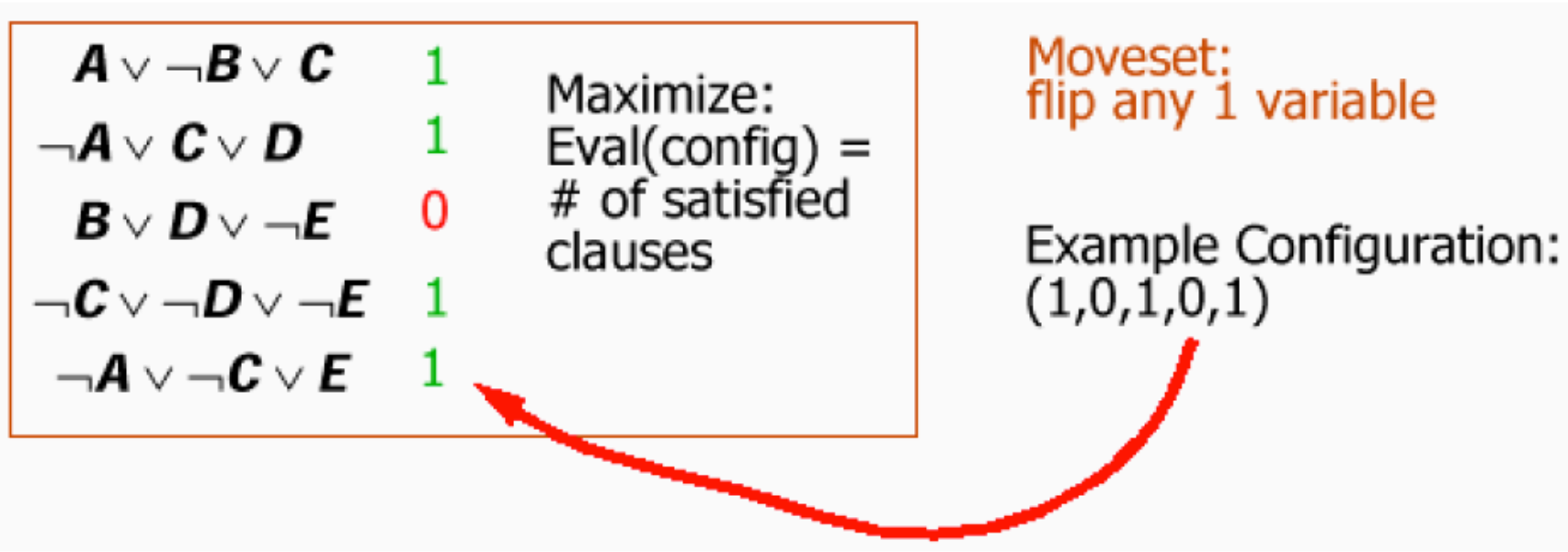
- Can be very effective
- Should be tried whenever hill climbing is used

# Variations of Hill Climbing

- We are still greedy!  Only willing to move upwards.
- Important observation in life:

| | | |
|---|---|---|
| Sometimes one needs to temporarily step back in order to move forward. | = | Sometimes one needs to move to an inferior neighbor in order to escape a local optimum. |

# Hill Climbing Example: SAT




WALKSAT (randomized GSAT):

Pick a random unsatisfied clause;
Consider 3 moves: flipping each variable.
If any improve Eval, accept the best.
If none improve Eval, then 50% of the time, pick the move that is the least bad; 50% of the time, pick a random one.

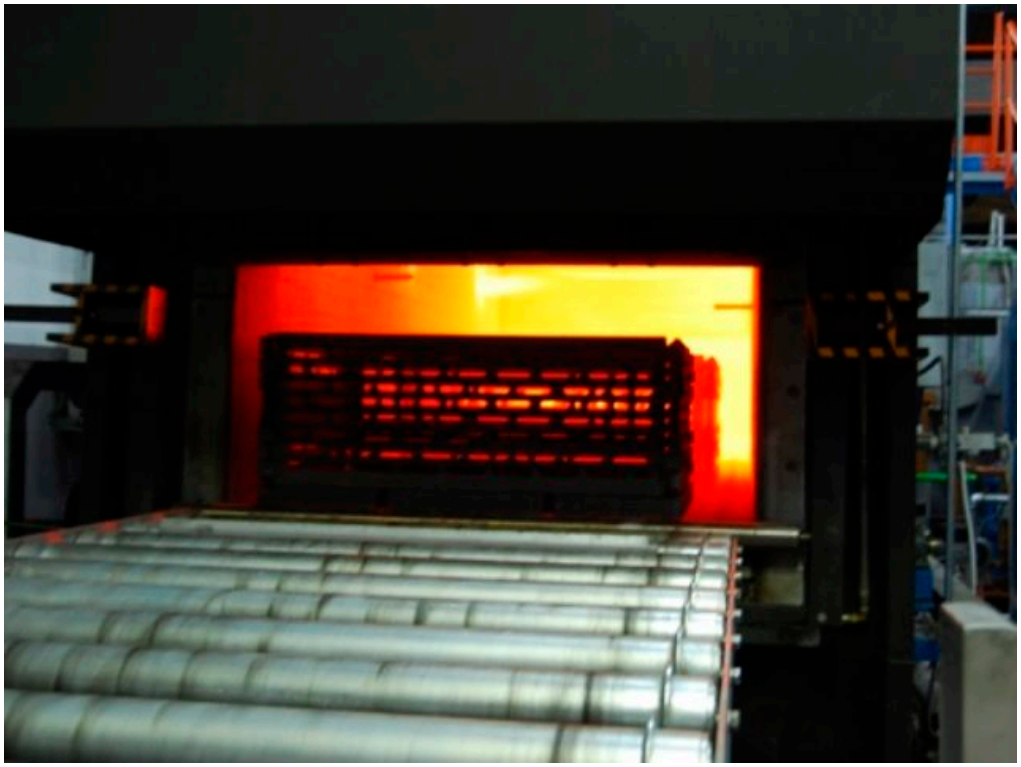
This is the best known algorithm for satisfying Boolean formulae! [Selman]

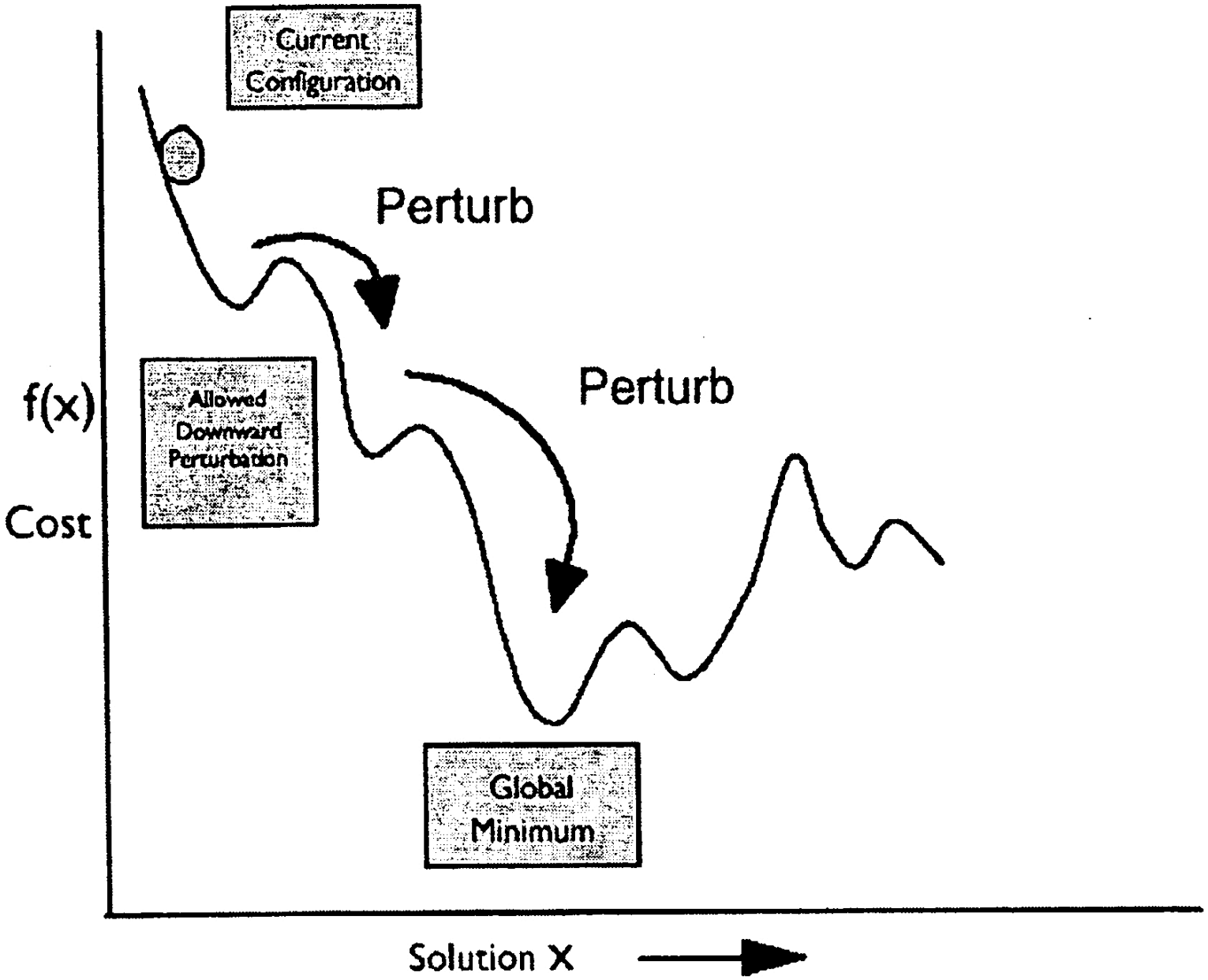
# 2. SIMULATED ANNEALING

# Simulated Annealing

**anneal**

- To subject (glass or metal) to a process of heating and slow cooling in order to toughen and reduce brittleness.






http://ashakhov.files.wordpress.com/2011/01/6725437-0-large.jpg

# Simulated Annealing

1. Let X := initial config
2. Let E := Eval(X)
3. Let i = random move from the moveset
4. Let $E_i$ := Eval(move(X,i))
5. If E < $E_i$ then
   X := move(X,i)
   E := $E_i$
   Else with some probability, accept the move even though things get worse:
   X := move(X,i)
   E := $E_i$
6. Goto 3 unless bored.

# Simulated Annealing

1. Let X := initial config
2. Let E := Eval(X)
3. Let i = random move from the moveset
4. Let $E_i$ := Eval(move(X,i))
5. If E < $E_i$ then
   X := move(X,i)
   E := $E_i$
   Else with some probability, accept the move even though things get worse:
   X := move(X,i)
   E := $E_i$
6. Goto 3 unless bored.

This may make the search fall out of mediocre local minima and into better local maxima.

How should we choose the probability of accepting a worsening move?

- *Idea One.* Probability = 0.1
- *Idea Two.* Probability decreases with time
- *Idea Three.* Probability decreases with time, and also as E – $E_i$ increases.

# Simulated Annealing

If $E_i >= E$ then definitely accept the change.

If $E_i < E$ then accept the change with probability

$$\mathbf{exp}\ (-(E - E_i)/T_i)$$

(called the Boltzman distribution)

…where $T_i$ is a "temperature" parameter that gradually decreases. Typical cooling schedule:

$T_i = T_0 \cdot r^i$

High temp: accept all moves (Random Walk)
Low temp: Stochastic Hill-Climbing

When enough iterations have passed without improvement, terminate.

This idea was introduced by Metropolis in 1953. It is "based" on "similarities" and "analogies" with the way that alloys manage to find a nearly global minimum energy level when they are cooled slowly.

# Aside: Analogy-based algorithms

Your lecturer predicts that for any natural phenomenon you can think of, there will be at least one AI research group that will have a combinatorial optimization algorithm "based" on "analogies" and "similarities" with the phenomenon. Here's the beginning of the list…

- Metal cooling annealing
- Evolution / Co-evolution / Sexual Reproduction
- Thermodynamics
- Societal Markets
- Management Hierarchies
- Ant/Insect Colonies
- Immune System
- Animal Behavior Conditioning
- Neuron / Brain Models
- Hill-climbing (okay, that's a stretch…)
- Particle Physics
- Inability of Elephants to Play Chess

# Simulated Annealing Issues

- MoveSet design is critical. This is the real ingenuity – not the decision to use simulated annealing.
- Evaluation function design often critical.
- Annealing schedule often critical.
- It's often cheaper to evaluate an incremental change of a previously evaluated object than to evaluate from scratch. Does simulated annealing permit that?
- What if approximate evaluation is cheaper than accurate evaluation?

# Simulated Annealing Discussions

Simulated annealing is sometimes empirically much better at avoiding local minima than hill-climbing. It is a successful, frequently-used, algorithm. Worth putting in your algorithmic toolbox.

Sadly, not much opportunity to say anything formal about it (though there is a proof that with an infinitely slow cooling rate, you'll find the global optimum).

There are mountains of practical, and problem-specific, papers on improvements.

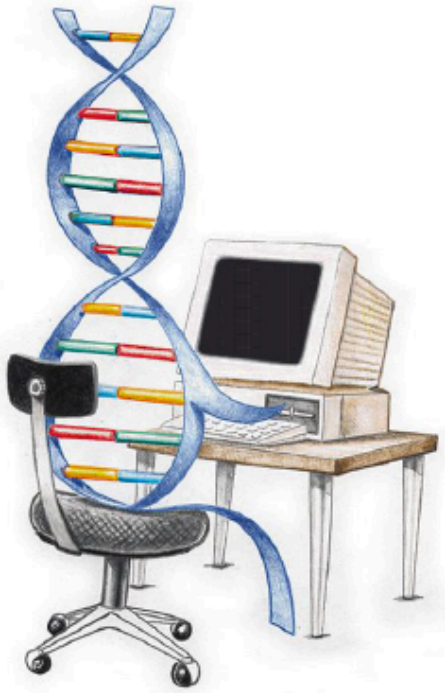
# SA for Minimization

http://www.youtube.com/watch?v=iaq_Fpr4KZc

# SA for TSP Demo

- http://www.youtube.com/watch?v=rsGOB80v0-k
- https://www.youtube.com/watch?v=RtqoGeXDDbQ&feature=youtube

# GENETIC ALGORITHM

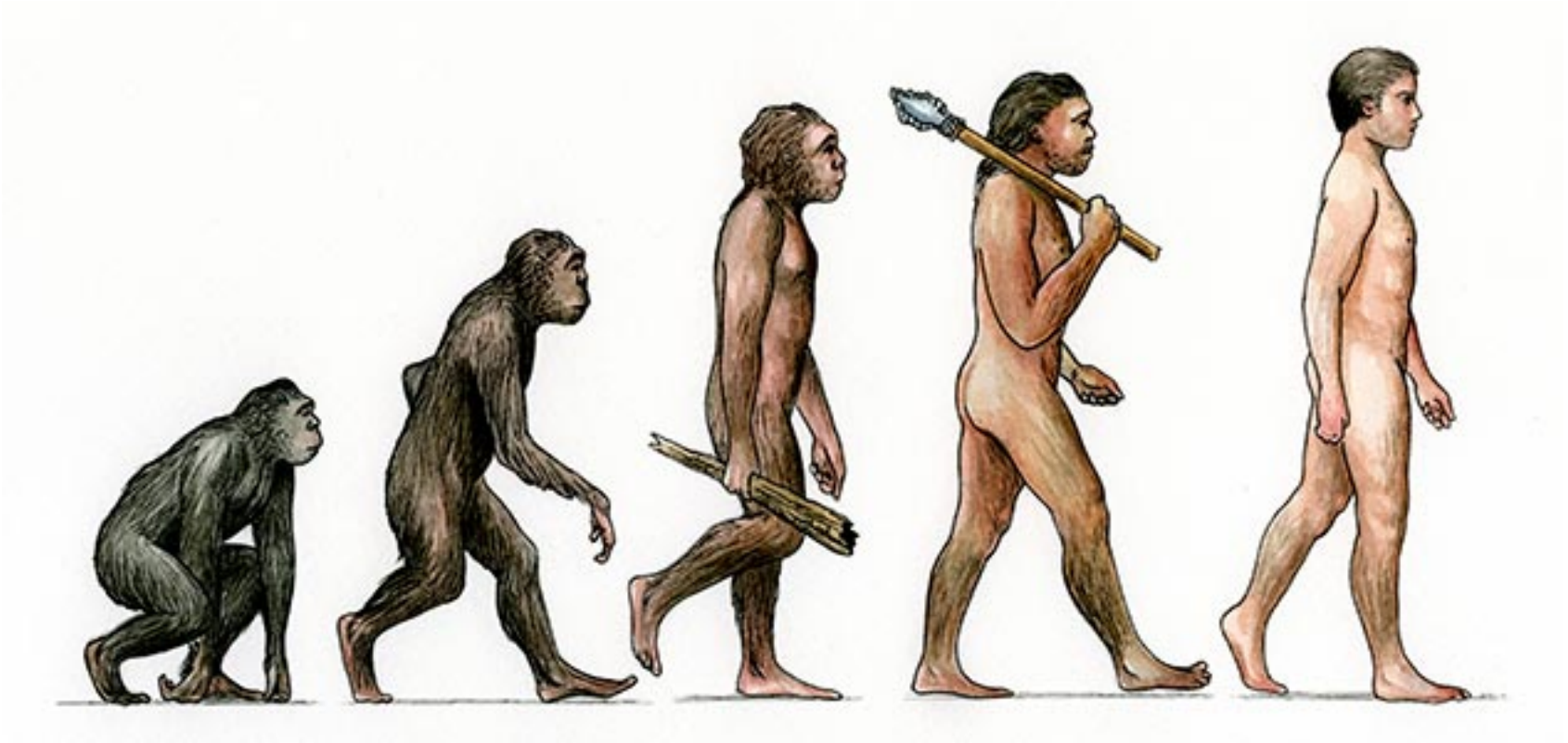


http://www.genetic-programming.org/

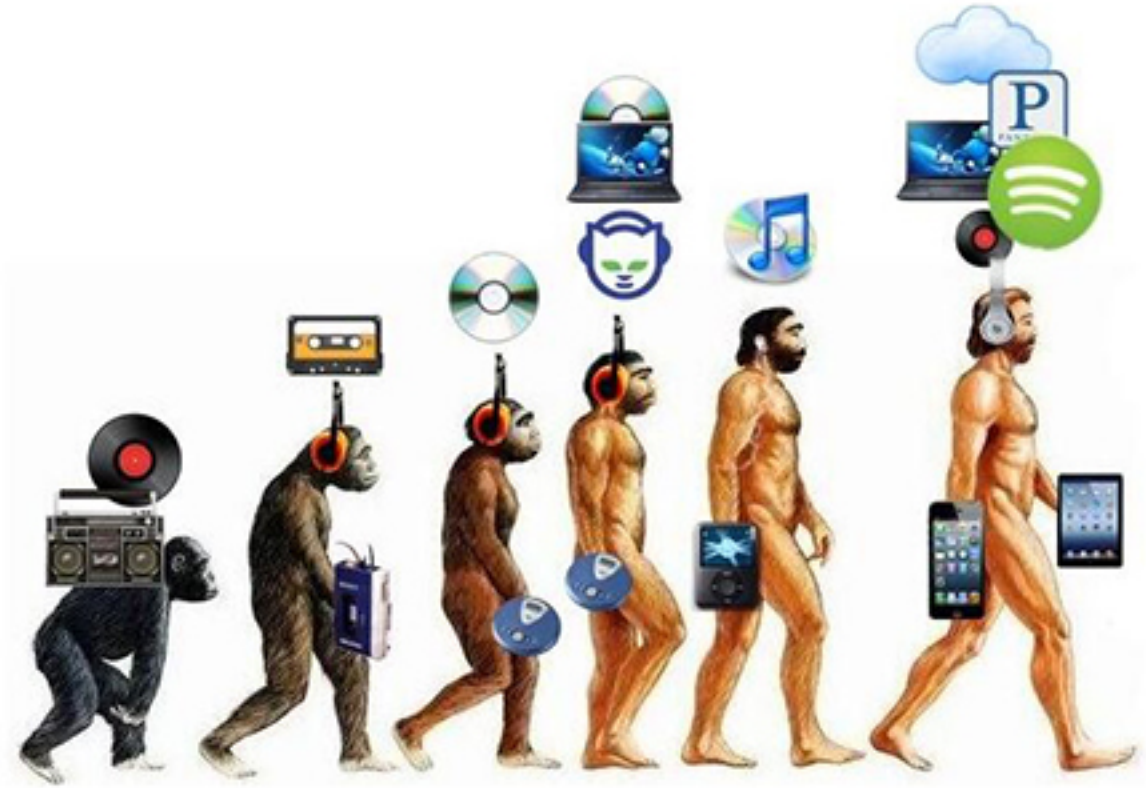
# Evolution

- Survival of the fittest, a.k.a. natural selection
- Genes encoded as DNA (deoxyribonucleic acid), sequence of bases: A (Adenine), C (Cytosine), T (Thymine) and G (Guanine)
- The chromosomes from the parents exchange randomly by a process called **crossover**. Therefore, the offspring exhibit some traits of the father and some traits of the mother.
  - Requires genetic diversity among the parents to ensure sufficiently varied offspring
- A rarer process called **mutation** also changes the genes (e.g. from cosmic ray).
  - Nonsensical/deadly mutated organisms die.
  - Beneficial mutations produce “stronger” organisms
  - Neither: organisms aren’t improved.

# Evolution of Man?



# Evolution of Technology

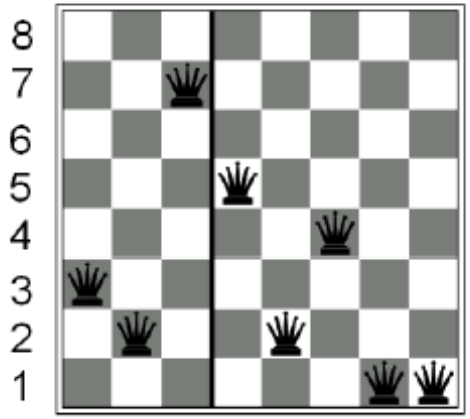


# Natural selection

- Individuals compete for resources
- Individuals with better genes have a larger chance to produce offspring, and vice versa
- After many generations, the population consists of lots of genes from the superior individuals, and less from the inferior individuals
- Superiority defined by fitness to the environment
- Popularized by Darwin
- Mistake of Lamarck: environment does not force an individual to change its genes

# Genetic algorithm

- Yet another AI algorithm based on real-world analogy
- Yet another heuristic stochastic search algorithm
- Each state *s* is called an individual. Often (carefully) coded up as a string.




(3 2 7 5 2 4 1 1)

- The score *f*(*s*) is called the fitness of *s*. Our goal is to find the global optimum (fittest) state.
- At any time we keep a fixed number of states. They are called the population. Similar to beam search.

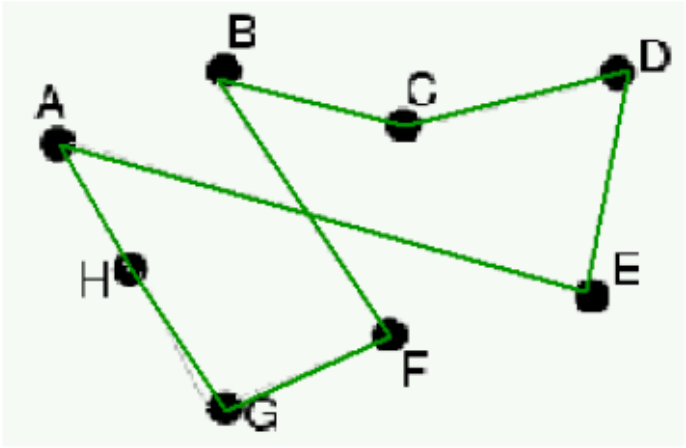
# Individual encoding

- The “DNA”
- Satisfiability problem

  (A B C D E) = (T F T T T)
- TSP

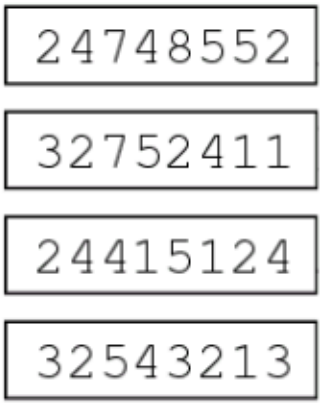
  A-E-D-C-B-F-G-H-A

$$A \vee \neg B \vee C$$
$$\neg A \vee C \vee D$$
$$B \vee D \vee \neg E$$
$$\neg C \vee \neg D \vee \neg E$$
$$\neg A \vee \neg C \vee E$$



# Genetic algorithm

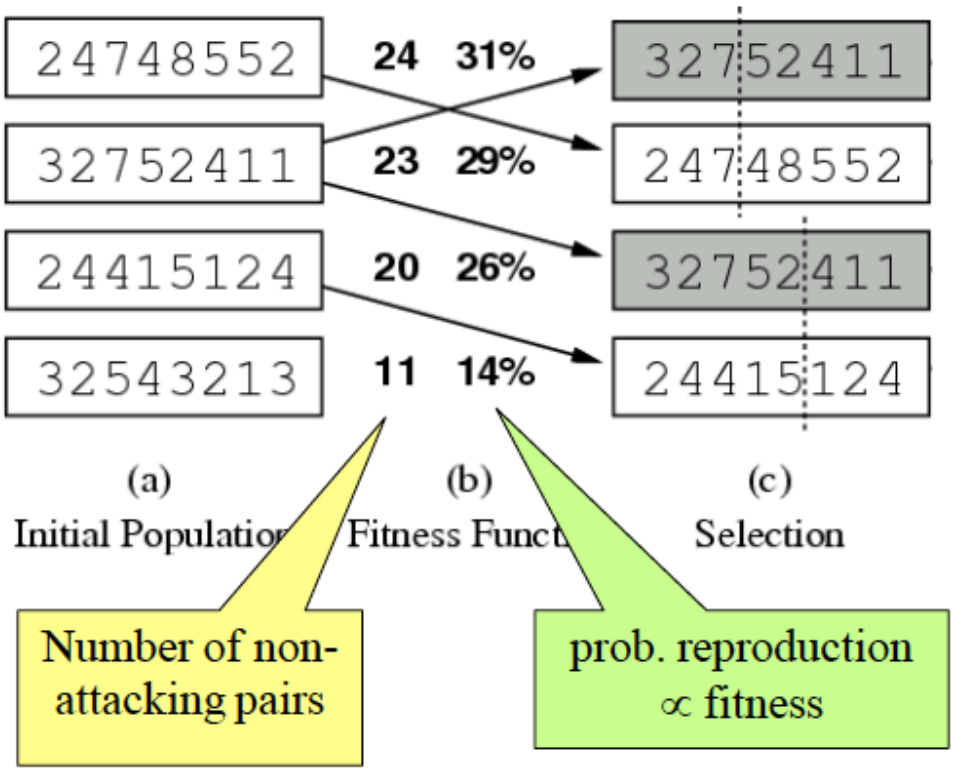
- **Genetic algorithm**: a special way to generate neighbors, using the analogy of cross-over, mutation, and natural selection.




(a)
Initial Population

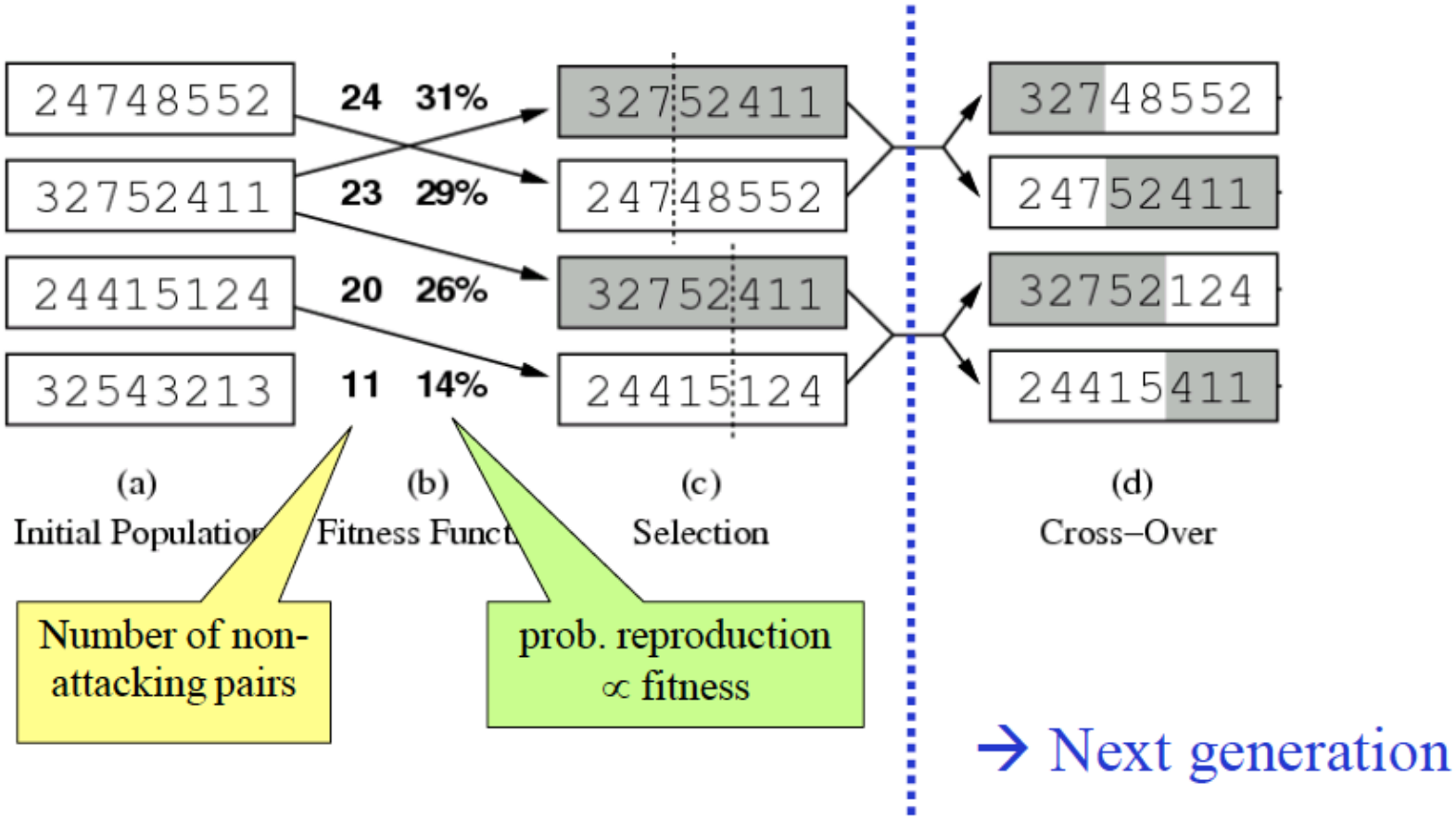
# Genetic algorithm

- **Genetic algorithm**: a special way to generate neighbors, using the analogy of cross-over, mutation, and natural selection.



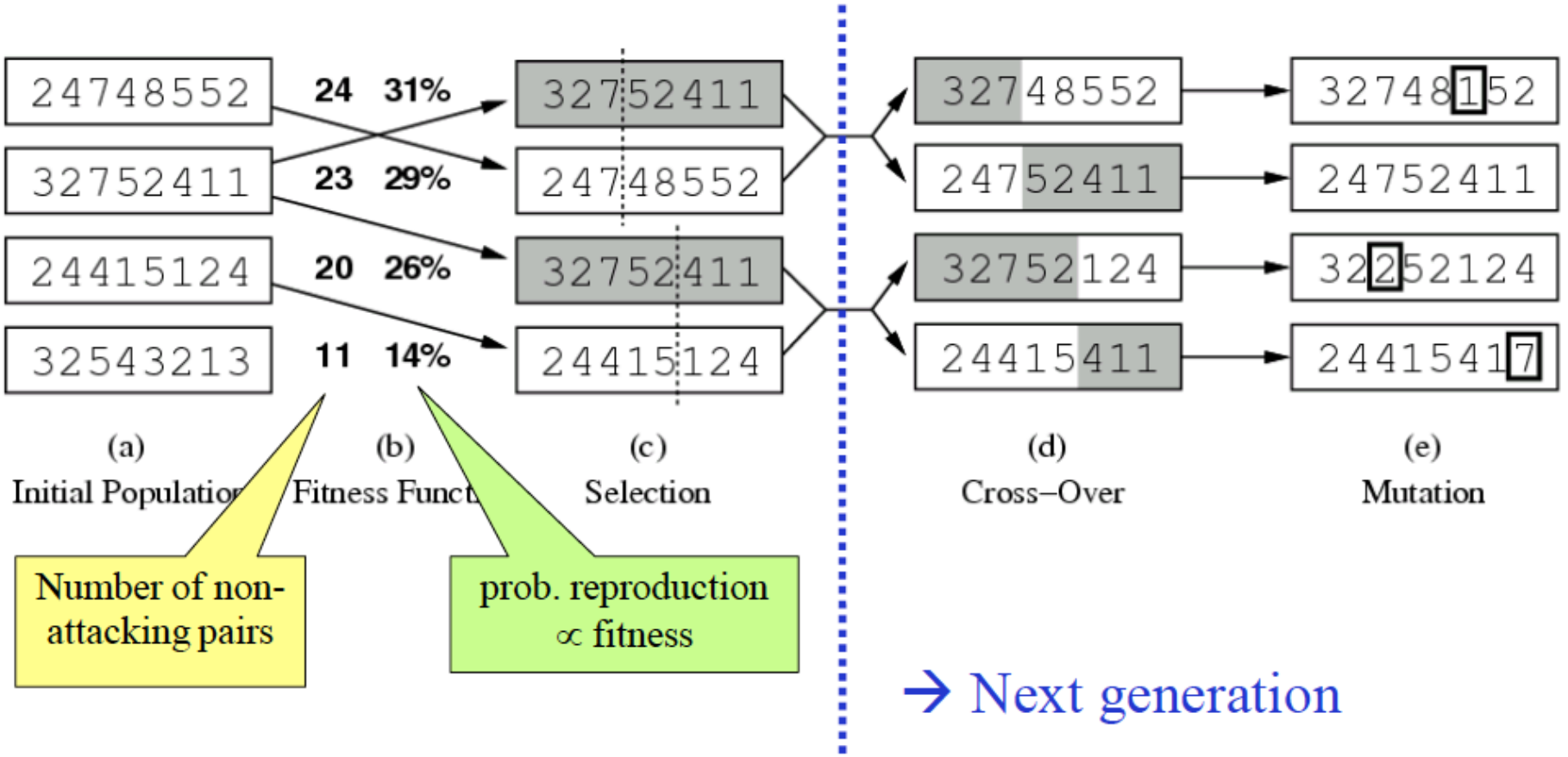
# Genetic algorithm

- **Genetic algorithm**: a special way to generate neighbors, using the analogy of cross-over, mutation, and natural selection.

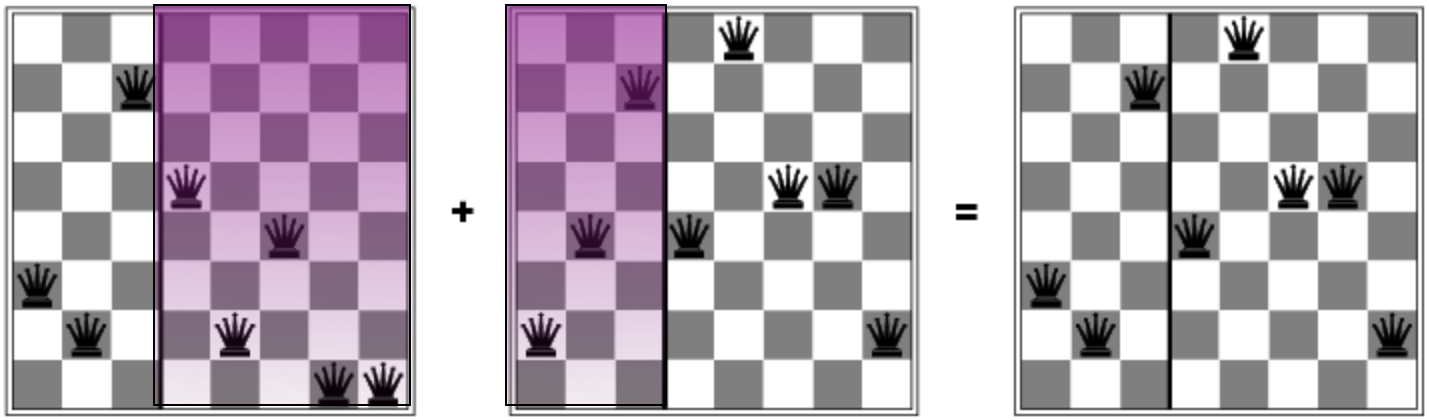


# Genetic algorithm

- **Genetic algorithm**: a special way to generate neighbors, using the analogy of cross-over, mutation, and natural selection.



# Chess Cross Over



Has the effect of “jumping” to a completely different new part of the search space (quite non-local)

# Genetic algorithm (one variety)

1. Let $s_1, \ldots, s_N$ be the current population
2. Let $p_i = f(s_i) / \Sigma_j f(s_j)$ be the reproduction probability
3. FOR $k = 1; k<N; k+=2$
   - parent1 = randomly pick according to $p$
   - parent2 = randomly pick another
   - randomly select a crossover point, swap strings of parents 1, 2 to generate children $t[k]$, $t[k+1]$
4. FOR $k = 1; k<=N; k++$
   - Randomly mutate each position in $t[k]$ with a small probability (mutation rate)
5. The new generation replaces the old: $\{ s \} \leftarrow \{ t \}$. Repeat.

# Proportional selection

- $p_i = f(s_i) / \Sigma_j f(s_j)$
- $\Sigma_j f(s_j) = 5+20+11+8+6=50$
- $p_1=5/50=10\%$

| Individual | Fitness | Prob. |
|---|---|---|
| A | 5 | 10% |
| B | 20 | 40% |
| C | 11 | 22% |
| D | 8 | 16% |
| E | 6 | 12% |

# Variations of genetic algorithm

- Parents may survive into the next generation
- Use ranking instead of $f(s)$ in computing the reproduction probabilities.
- Cross over random bits instead of chunks.
- Optimize over sentences from a programming language. Genetic programming.
- …

# Genetic algorithm issues

- State encoding is the real ingenuity, not the decision to use genetic algorithm.
- Lack of diversity can lead to premature convergence and non-optimal solution
- Not much to say theoretically
  - Cross over (sexual reproduction) much more efficient than mutation (asexual reproduction).
- Easy to implement.
- Try hill-climbing with random restarts first!

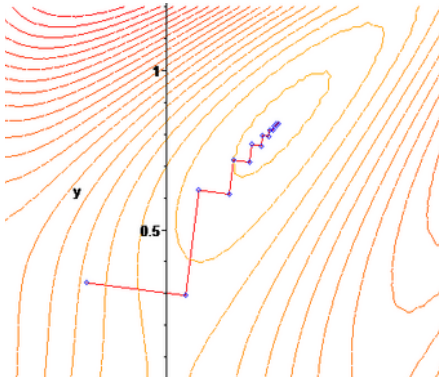
# Demo of GA

- http://www.youtube.com/watch?v=uxourrlPlf8

# Relation Between Iterative Improvement and Other Methods

# Gradient Descent

Assume we have some cost-function: $C(x_1,\ldots,x_n)$
and we want minimize over continuous variables X1,X2,..,Xn




1. Compute the *gradient :* $\frac{\partial}{\partial x_i} C(x_1,\ldots,x_n) \qquad \forall i$

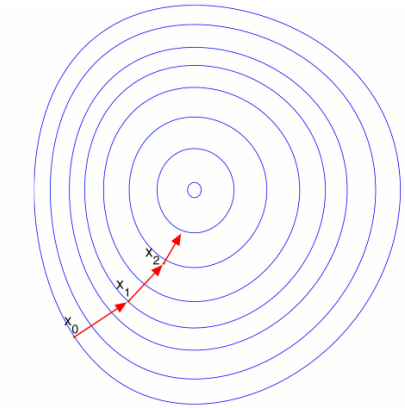
2. Take a small step downhill in the direction of the gradient:

$$x_i \rightarrow x'_i = x_i - \lambda \frac{\partial}{\partial x_i} C(x_1,\ldots,x_n) \qquad \forall i$$

3. Check if $C(x_1,..,x'_i,..,x_n) < C(x_1,..,x_i,..,x_n)$

4. If true then accept move, if not reject.

5. Repeat.




Smyth, 2007

# Learning as optimization

- Many machine learning problems can be casted as optimization

- Example:
  - Training data D = $\{(\underline{x}_1, c_1), \ldots\ldots (\underline{x}_n, c_n)\}$
    where $\underline{x}_i$ = feature or attribute vector
    and $c_i$ = class label (say binary-valued)

  - We have a model (a function or classifier) that maps from x to c
    e.g., sign( $\underline{w}.\underline{x}'$ ) = {-1, +1}

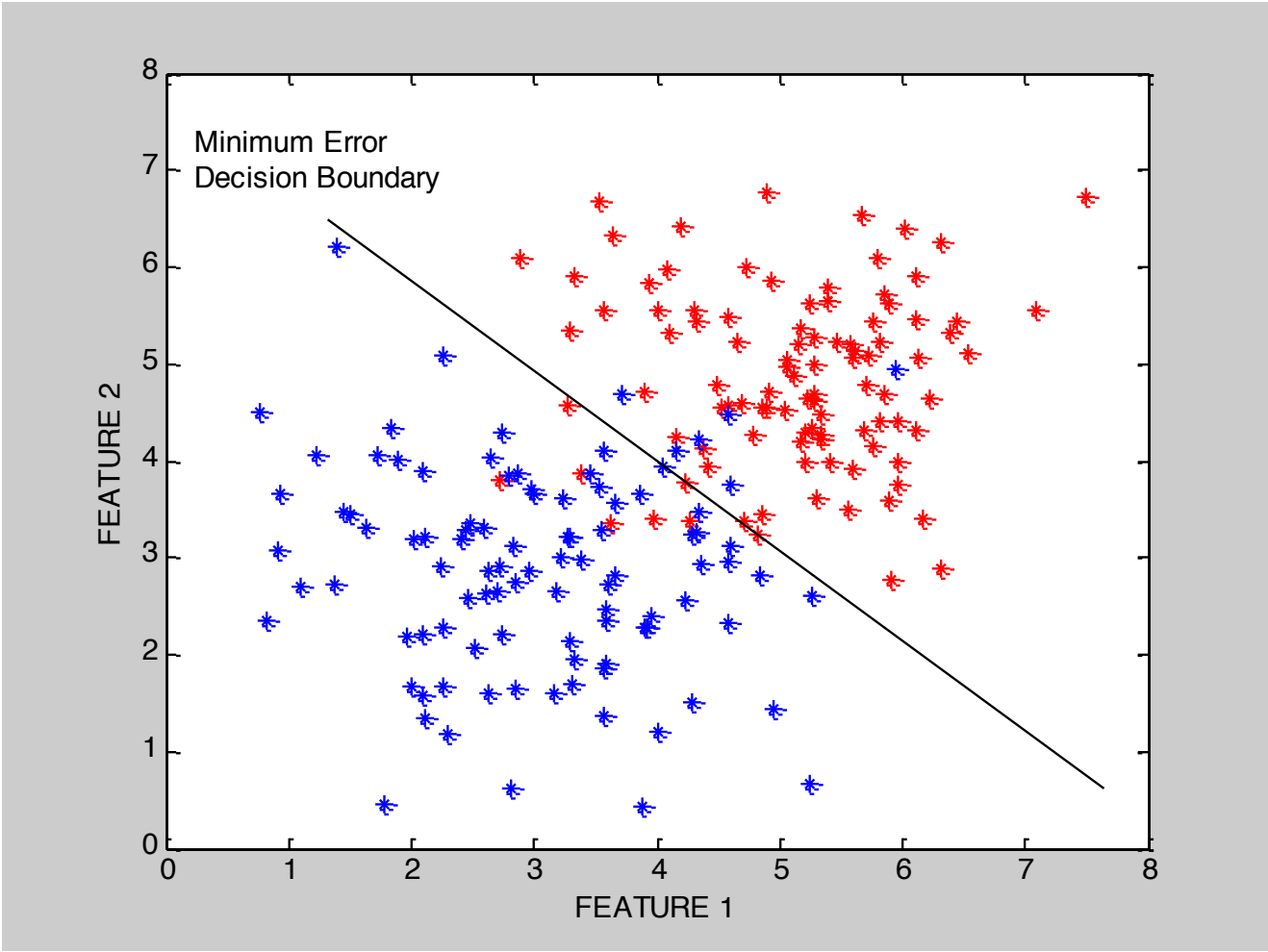
  - We can measure the error E($\underline{w}$) for any setting of the weights $\underline{w}$, and given a training data set D

  - Optimization problem: find the weight vector that minimizes $\underline{E(w)}$

(general idea is "empirical error minimization")

Smyth, 2007

# Learning a minimum error decision boundary




Smyth, 2007