

Advanced Search

Hill climbing, simulated annealing, genetic algorithm

Lecturer: Ji Liu

Thank Jerry Zhu for sharing slides

[Based on slides from Andrew Moore <http://www.cs.cmu.edu/~awm/tutorials>]

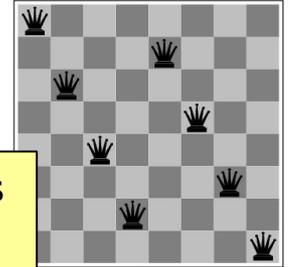
Optimization problems

- Previously we want a **path** from start to goal
 - **Uninformed search**: $g(s)$: Iterative Deepening
 - **Informed search**: $g(s)+h(s)$: A*
- **Now a different setting**:
 - Each state s has a **score** $f(s)$ that we can compute
 - The goal is to find the state with the **highest score**, or a **reasonably high score**
 - Do not care about the path
 - This is an **optimization problem**
 - Enumerating the states is intractable
 - Even previous search algorithms are too expensive

Examples

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

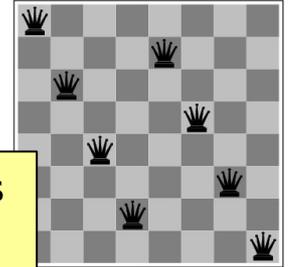
Note we want s with the lowest score $f(s)=0$. The techniques are the same. Low or high should be obvious from context.



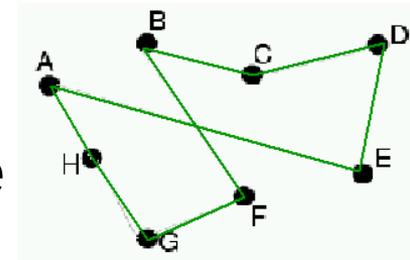
Examples

- 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.



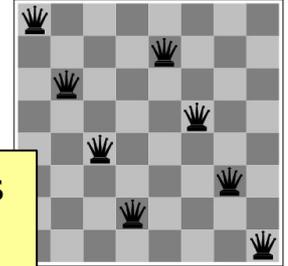
- Traveling salesperson problem (TSP)
 - Visit each city once, return to first city
 - State = order of cities, $f(s)$ = total mileage



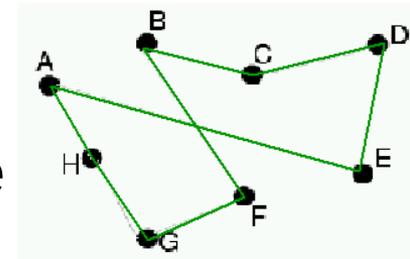
Examples

- 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



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

1. HILL CLIMBING

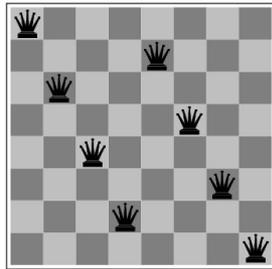


Hill climbing

- 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

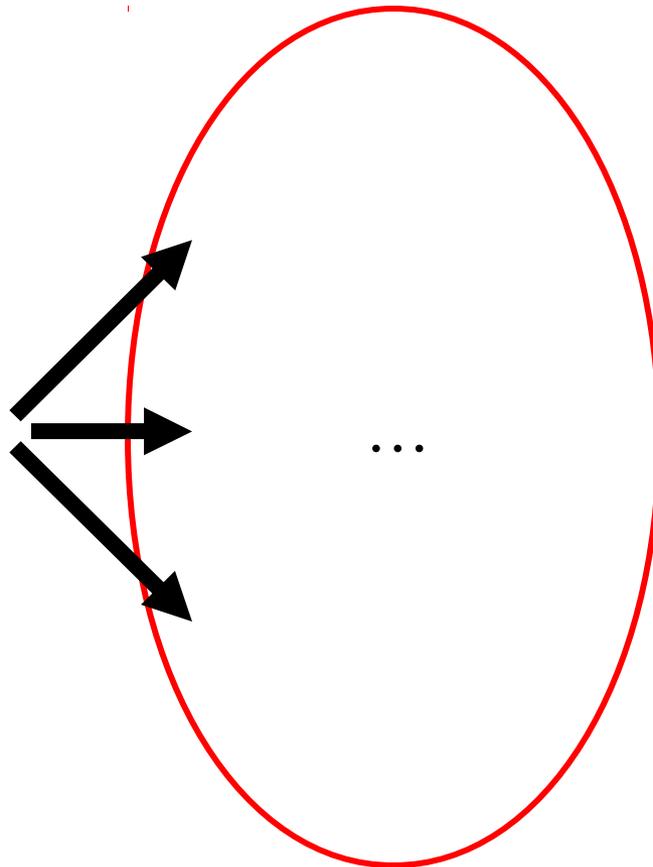
Neighbors: N-queen

- Example: N-queen (one queen per column). One possibility:



s

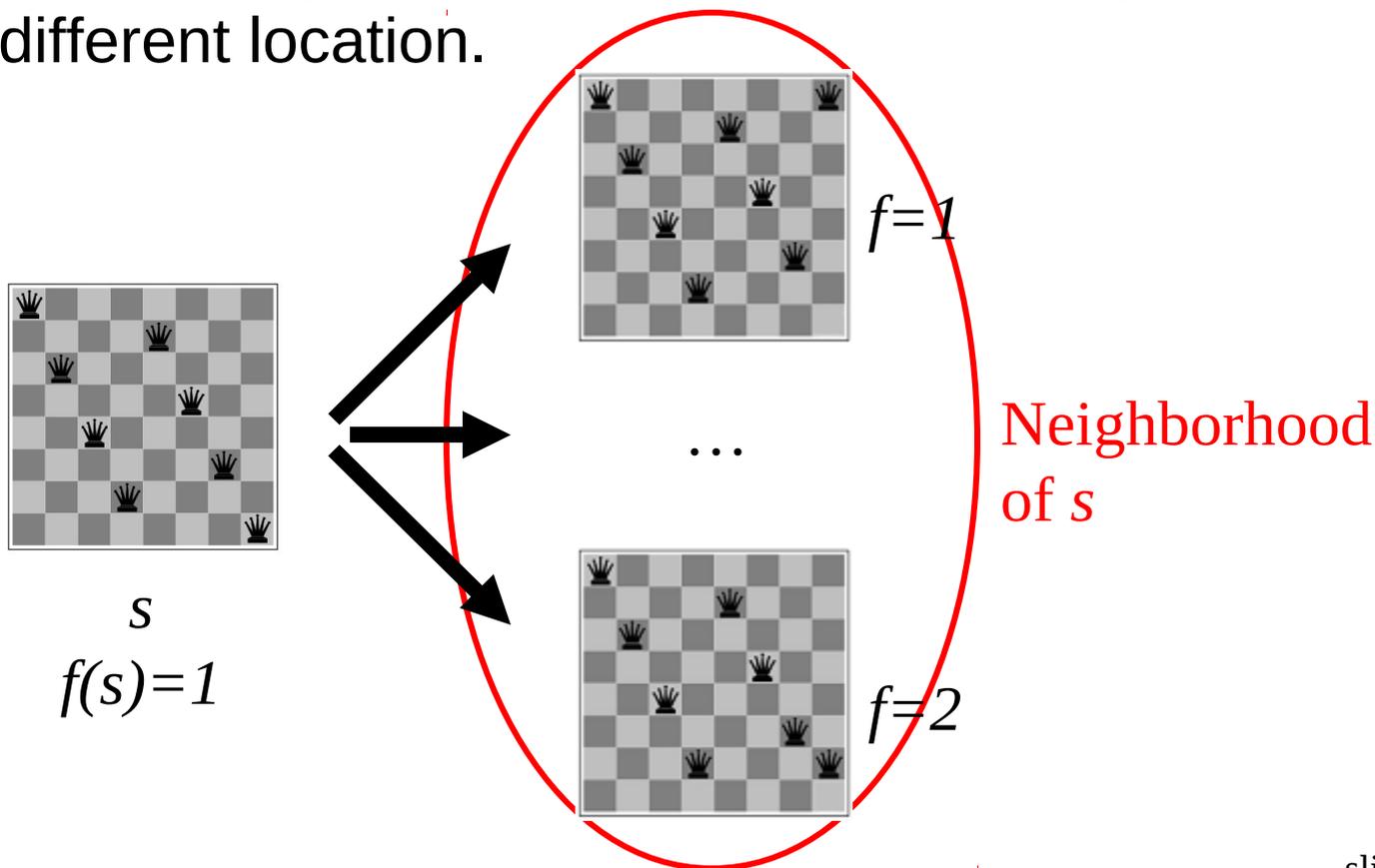
$$f(s)=1$$



Neighborhood
of s

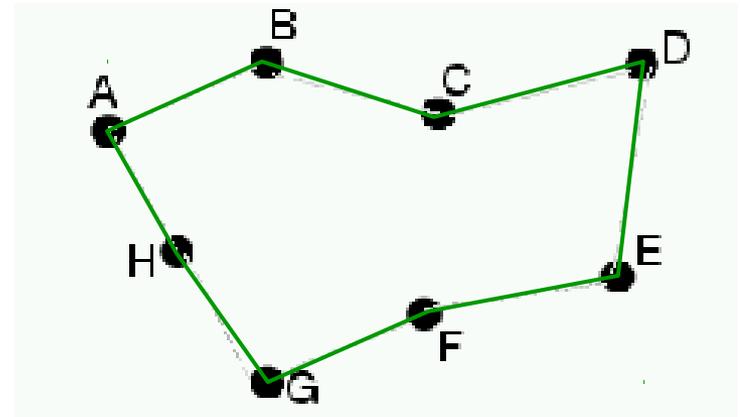
Neighbors: N-queen

- 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.



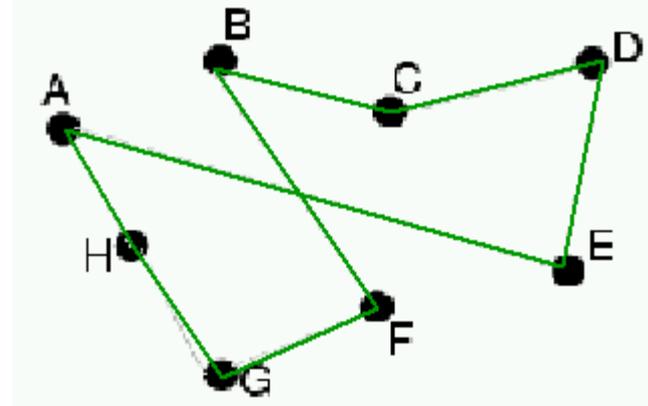
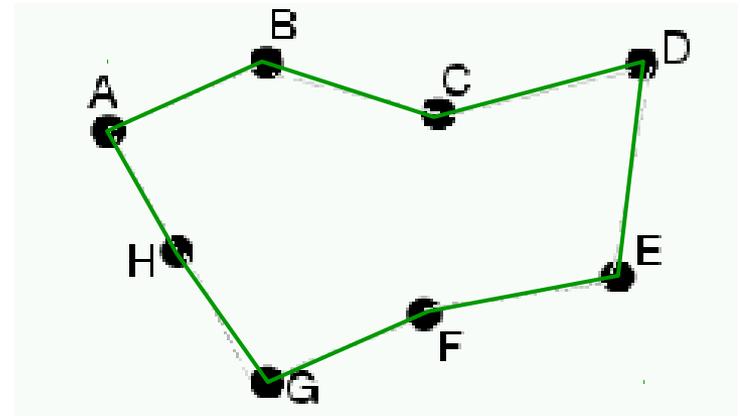
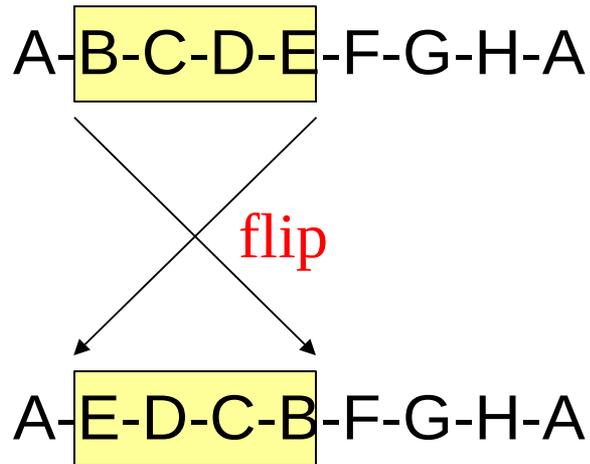
Neighbors: TSP

- state: A-B-C-D-E-F-G-H-A
- f = length of tour



Neighbors: TSP

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



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

1. Pick initial state s
2. Pick t in neighbors(s) with the largest $f(t)$
3. IF $f(t) \leq f(s)$ THEN stop, return s
4. $s = t$. GOTO 2.

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

Hill climbing algorithm

1. Pick initial state s
2. Pick t in neighbors(s) with the largest $f(t)$
3. IF $f(t) \leq f(s)$ THEN stop, return s
4. $s = t$. GOTO 2.

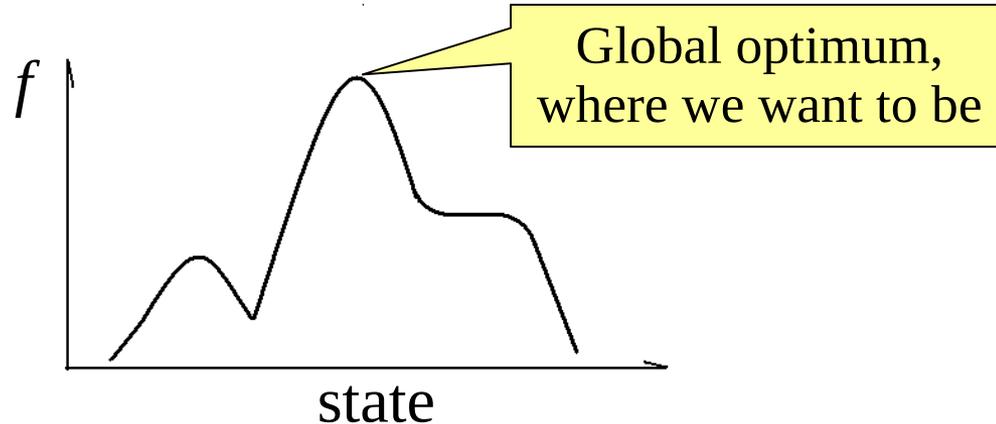
- Not the most sophisticated algorithm in the world.
- Very greedy.
- Easily stuck.

your enemy:

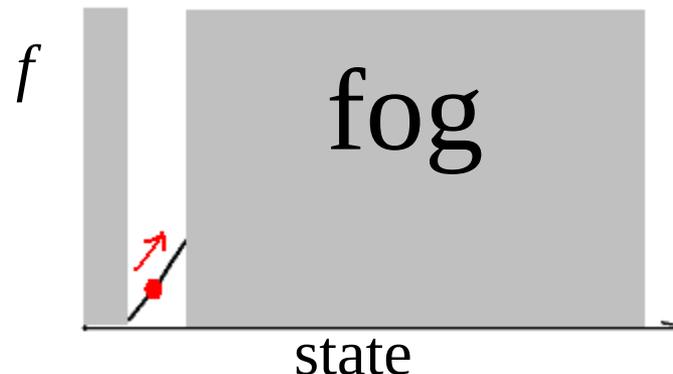
**local
optima**

Local optima in hill climbing

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

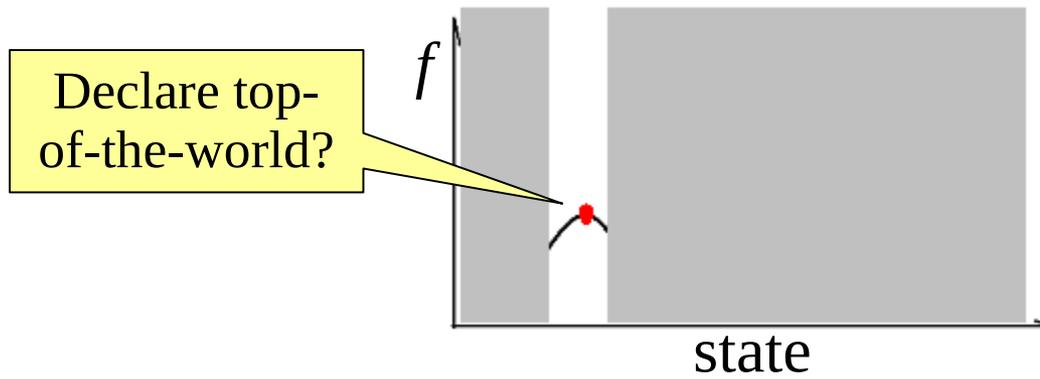


- But we can't see the landscape all at once. Only see the neighborhood. Climb in fog.

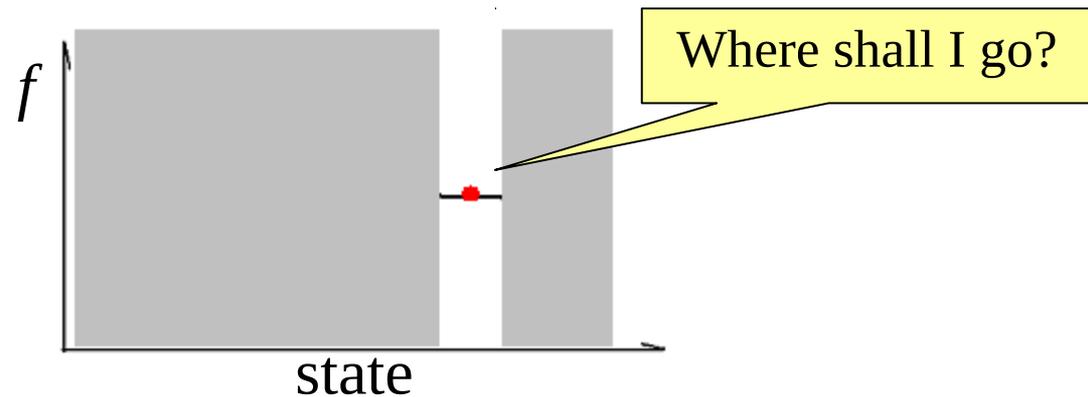


Local optima in hill climbing

- Local optima (there can be many!)



- Plateaux



Local optima in hill climbing

- Local optima (there can be many)

Declare to
the world

The rest of the lecture is
about

**Escaping
local optima**

- Plateaus

Where shall I go?

Not every local minimum should be escaped



Variation 1: hill climbing with random restarts

- 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 2 and 3 of hill climbing

- **Question:** How do we make hill climbing less greedy?
 - Stochastic hill climbing
 - Randomly select among better neighbors
 - The better, the more likely
 - Pros / cons compared with basic hill climbing?

Variations 2 and 3 of hill climbing

- **Question:** How do we make hill climbing less greedy?
 - Stochastic hill climbing
 - Randomly select among better neighbors
 - The better, the more likely
 - Pros / cons compared with basic hill climbing?
- **Question:** What if the neighborhood is too large to enumerate? (e.g. N-queen if we need to pick both the column and the move within it)

Variations 2 and 3 of hill climbing

- **Question:** How do we make hill climbing less greedy?
 - Stochastic hill climbing
 - Randomly select among better neighbors
 - The better, the more likely
 - Pros / cons compared with basic hill climbing?
- **Question:** What if the neighborhood is too large to enumerate? (e.g. N-queen if we need to pick both the column and the move within it)
 - First-choice hill climbing
 - Randomly generate neighbors, one at a time
 - If better, take the move
 - Pros / cons compared with basic hill climbing?

Variation 4 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.

Variation 4 of hill climbing

WALKSAT [Selman]

- Consider 3 neighbors
- If any improves f , accept the best
- If none improves f :
 - 50% of the time pick the least bad neighbor
 - 50% of the time pick a random neighbor

This is the best known algorithm for satisfying Boolean formulae.

$$\begin{array}{l} A \vee \neg B \vee C \\ \neg A \vee C \vee D \\ B \vee D \vee \neg E \\ \neg C \vee \neg D \vee \neg E \\ \neg A \vee \neg C \vee E \end{array}$$



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.

Simulated Annealing

1. Pick initial state s
2. Randomly pick t in neighbors(s)
3. IF $f(t)$ better THEN accept $s \leftarrow t$.
4. ELSE /* t is worse than s */
5. accept $s \leftarrow t$ with a small probability
6. GOTO 2 until bored.

Simulated Annealing

1. Pick initial state s
2. Randomly pick t in neighbors(s)
3. IF $f(t)$ better THEN accept $s \leftarrow t$.
4. ELSE /* t is worse than s */
5. accept $s \leftarrow t$ with a small probability
6. GOTO 2 until bored.

How to choose the small probability?

idea 1: $p = 0.1$

Simulated Annealing

1. Pick initial state s
2. Randomly pick t in neighbors(s)
3. IF $f(t)$ better THEN accept $s \leftarrow t$.
4. ELSE /* t is worse than s */
5. accept $s \leftarrow t$ with a small probability
6. GOTO 2 until bored.

How to choose the small probability?

idea 1: $p = 0.1$

idea 2: p decreases with time

Simulated Annealing

1. Pick initial state s
2. Randomly pick t in neighbors(s)
3. IF $f(t)$ better THEN accept $s \leftarrow t$.
4. ELSE /* t is worse than s */
5. accept $s \leftarrow t$ with a small probability
6. GOTO 2 until bored.

How to choose the small probability?

idea 1: $p = 0.1$

idea 2: p decreases with time

idea 3: p decreases with time, also as the 'badness'
 $|f(s) - f(t)|$ increases

Simulated Annealing

- If $f(t)$ better than $f(s)$, always accept t
- Otherwise, accept t with probability

$$\exp\left(-\frac{|f(s) - f(t)|}{Temp}\right)$$



Boltzmann
distribution

Simulated Annealing

- If $f(t)$ better than $f(s)$, always accept t
- Otherwise, accept t with probability

$$\exp\left(-\frac{|f(s) - f(t)|}{Temp}\right)$$

Boltzmann
distribution

- $Temp$ is a temperature parameter that 'cools' (anneals) over time, e.g. $Temp \leftarrow Temp * 0.9$ which gives $Temp = (T_0)^{\#iteration}$
 - High temperature: almost always accept any t
 - Low temperature: first-choice hill climbing
- If the 'badness' (formally known as energy difference) $|f(s) - f(t)|$ is large, the probability is small.

SA algorithm

// assuming we want to maximize $f()$

current = Initial-State(problem)

for $t = 1$ **to** ∞ **do**

$T = \text{Schedule}(t)$; // T is the current temperature, which is monotonically decreasing with t

if $T=0$ **then return** current ; //halt when temperature = 0

next = Select-Random-Successor-State(current)

$\text{delta}E = f(\text{next}) - f(\text{current})$; // If positive, next is better than current. Otherwise, next is worse than current.

if $\text{delta}E > 0$ **then** current = next ; // always move to a better state

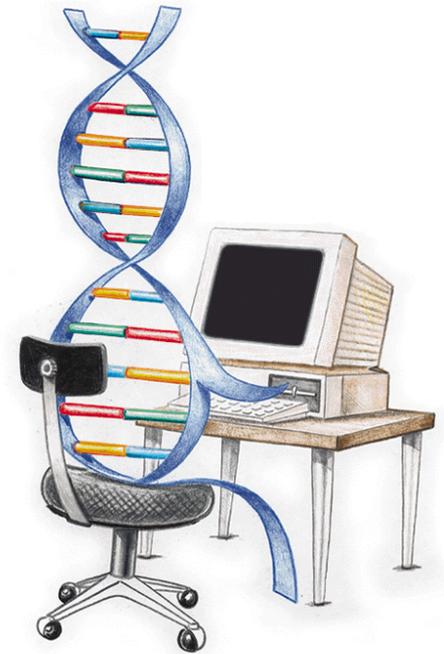
else current = next with probability $p = \exp(\text{delta}E / T)$; // as $T \rightarrow 0$, $p \rightarrow 0$; as $\text{delta}E \rightarrow -\infty$, $p \rightarrow 0$

end

Simulated Annealing issues

- Cooling scheme important
- Neighborhood design is the real ingenuity, not the decision to use simulated annealing.
- Not much to say theoretically
 - With infinitely slow cooling rate, finds global optimum with probability 1.
- Proposed by **Metropolis** in 1953 based on the analogy that alloys manage to find a near global minimum energy state, when annealed slowly.
- Easy to implement.
- Try hill-climbing with random restarts first!

GENETIC ALGORITHM



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

slide 38

Evolution

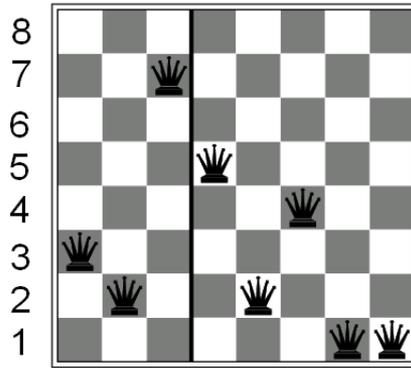
- Survival of the fittest, a.k.a. natural selection
- Genes encoded as DNA (deoxyribonucleic acid), sequence of bases: A (Adenine), C (Cytosine), T (Thymine) and G (Guanine)
- The chromosomes from the parents exchange randomly by a process called **crossover**. Therefore, the offspring exhibit some traits 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.

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.

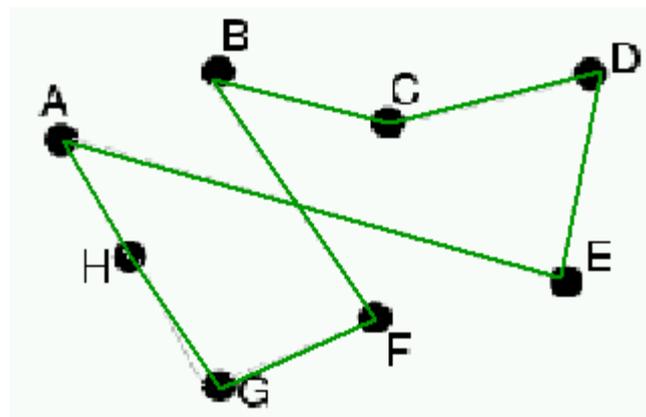


(3 2 7 5 2 4 1 1)

- 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

$$\begin{aligned} &A \vee \neg B \vee C \\ &\neg A \vee C \vee D \\ &B \vee D \vee \neg E \\ &\neg C \vee \neg D \vee \neg E \\ &\neg A \vee \neg C \vee E \end{aligned}$$


Genetic algorithm

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

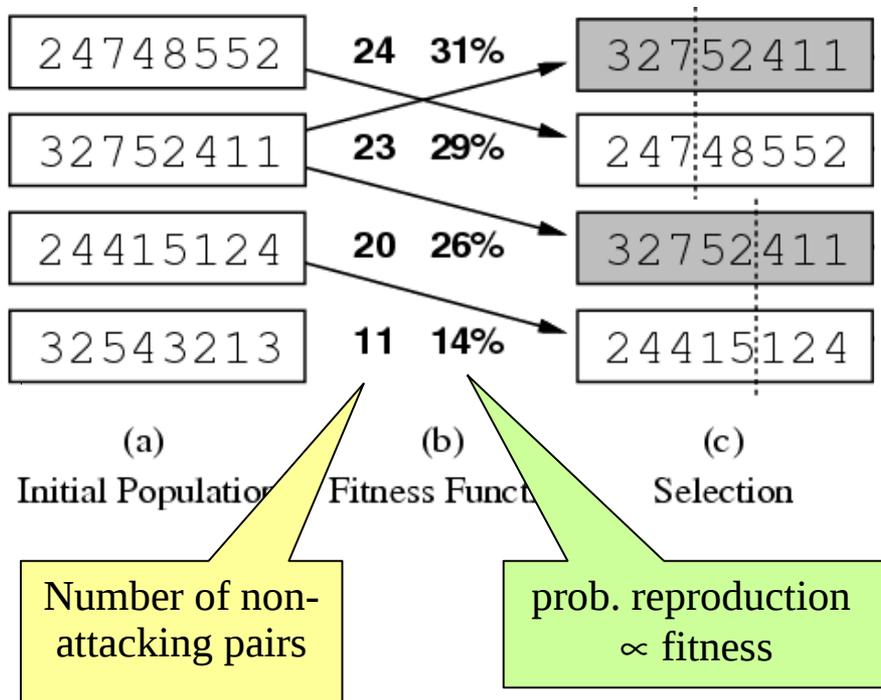
24748552
32752411
24415124
32543213

(a)

Initial Population

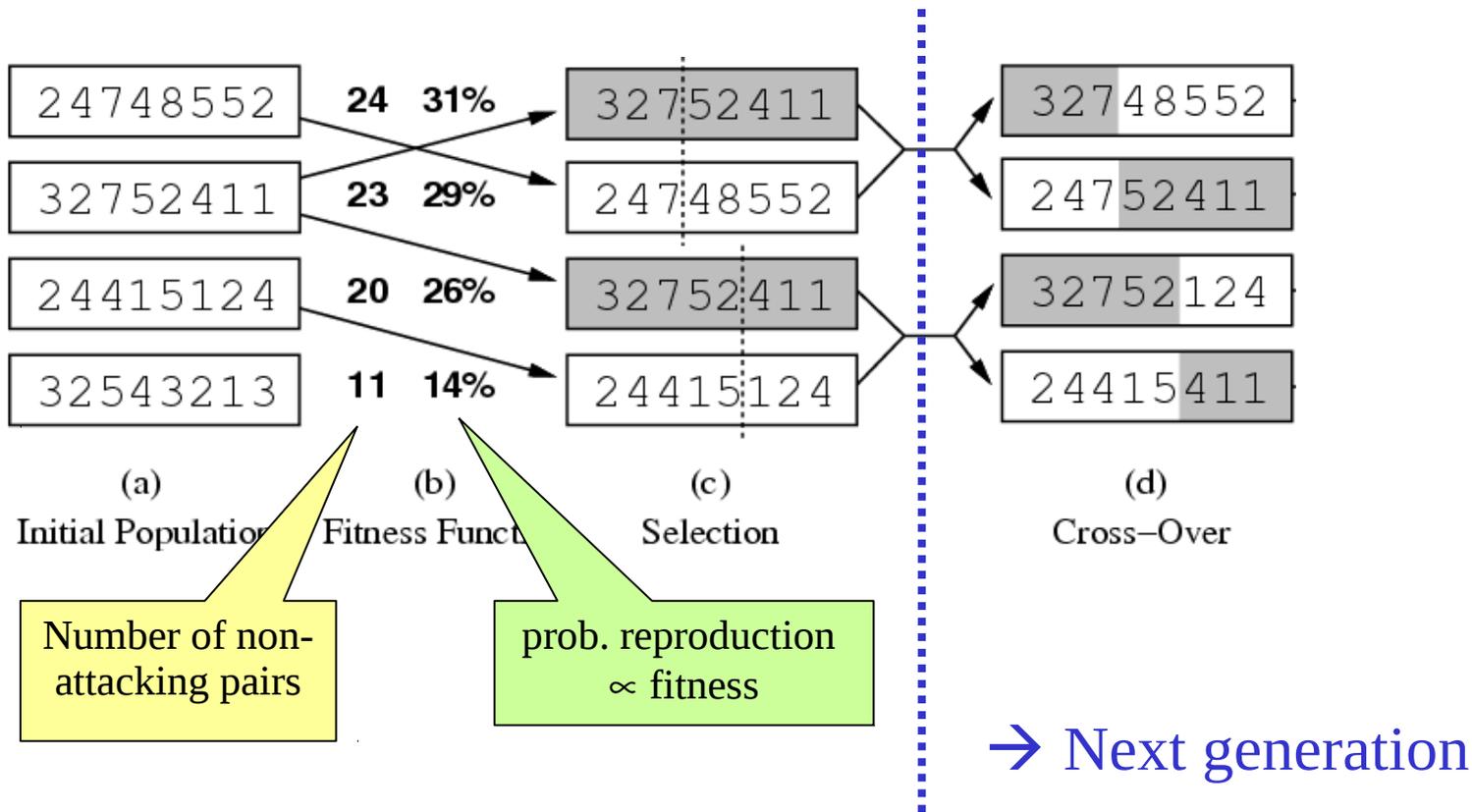
Genetic algorithm

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



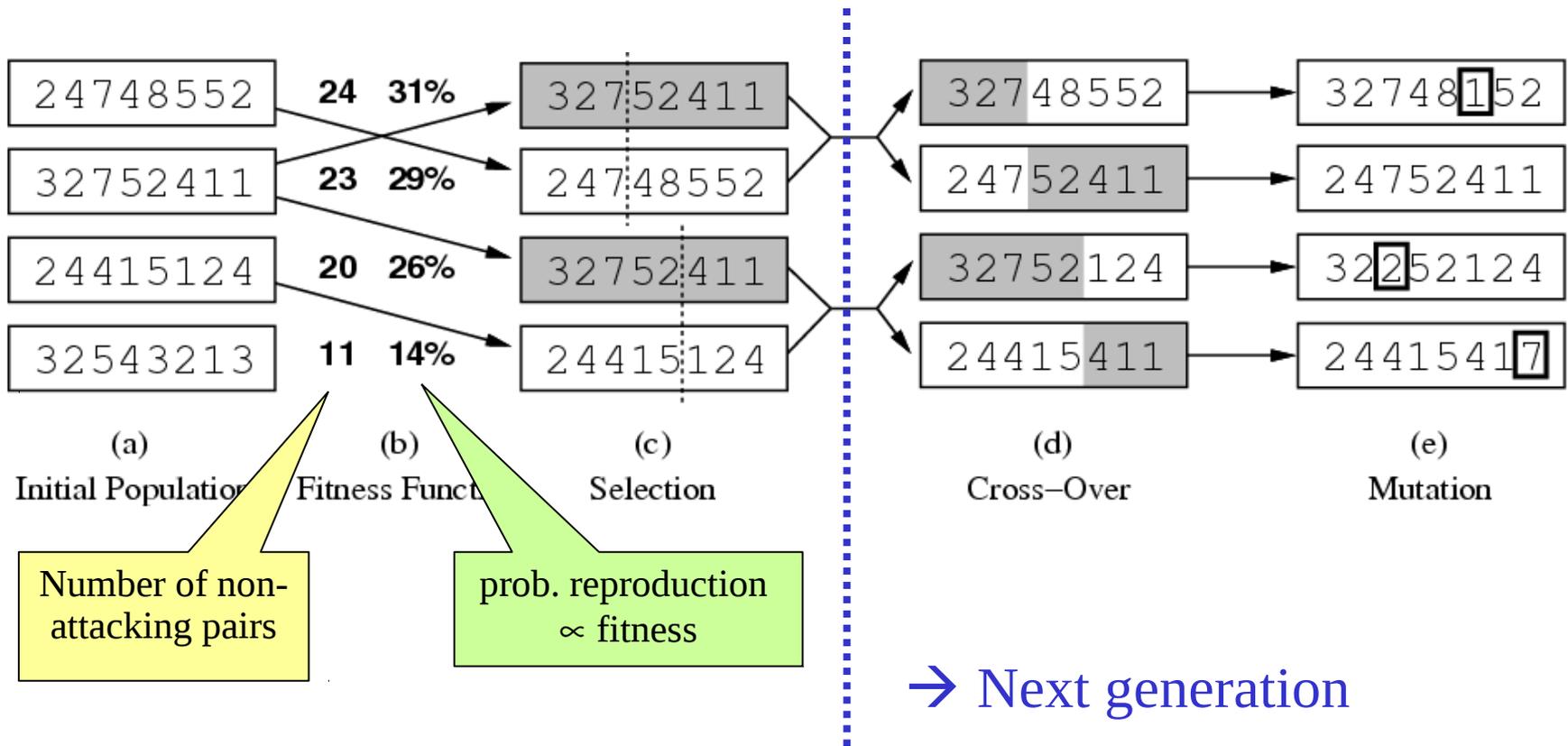
Genetic algorithm

- **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**.



Genetic algorithm (one variety)

1. Let s_1, \dots, s_N be the current population
2. Let $p_i = f(s_i) / \sum_j f(s_j)$ 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 $k = 1; k \leq N; k++$
 - 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

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

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!