# CS 540 Introduction to Artificial Intelligence

## **Search III: Advanced Search**

University of Wisconsin-Madison

**Fall 2022**

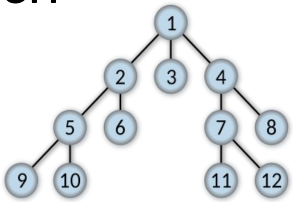
# Outline

- Advanced Search & Hill-climbing
  - More difficult problems, basics, local optima, variations
- Simulated Annealing
  - Basic algorithm, temperature, tradeoffs
- Genetic Algorithms
  - Basics of evolution, fitness, natural selection

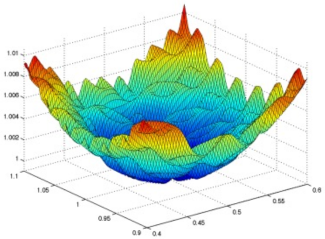
# Search vs. Optimization

Before: wanted a **path** from start state to goal state

- Uninformed search, informed search



Wiki


TuringFin

**New setting**: optimization

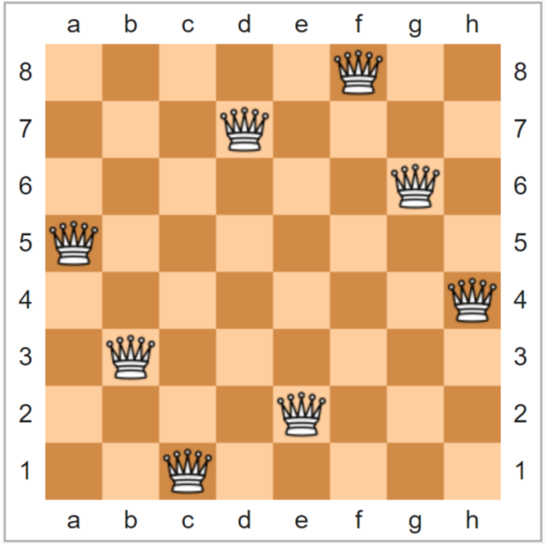
- States *s* have values ***f***(*s*)
- Want: *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)
- Challenging setting: **too many states** for previous search approaches, but maybe not a differentiable function for SGD.

# Examples: *n* Queens

A classic puzzle:

- Place 8 queens on a 8 x 8 chessboard so that no two have same row, column, or diagonal.
- Can generalize to n x n chessboard.

- What are states *s*? Values ***f***(*s*)?
  - State: configuration of the board
  - ***f***(*s*): # of conflicting queens



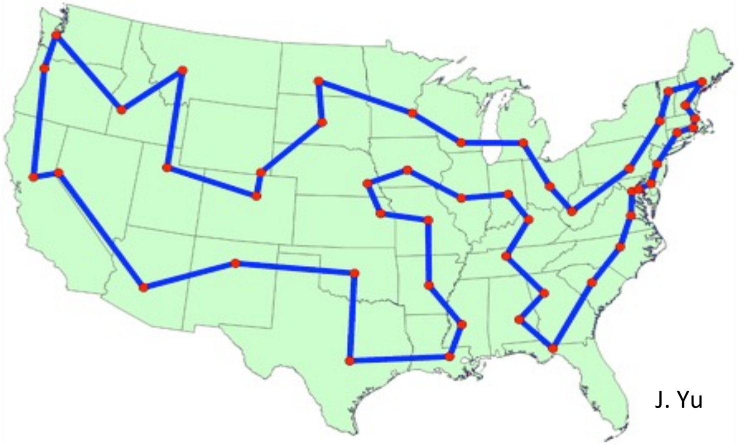

Wiki

# Examples: TSP

Famous graph theory problem.

- Get a graph G = (V,E). **Goal**: a path that visits each node exactly once and returns to the initial node (a **tour**).
  - State: a particular tour (i.e., ordered list of nodes)
  - *f*(*s*): total weight of the tour

  (e.g., total miles traveled)



J. Yu

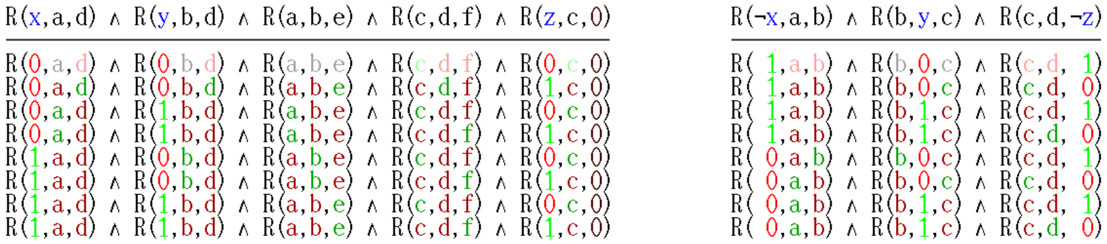
# Examples: Satisfiability

## Boolean satisfiability (e.g., 3-SAT)

- Recall our logic lecture. Conjunctive normal form

$$(A \vee \neg B \vee C) \wedge (\neg A \vee C \vee D) \wedge (B \vee D \vee \neg E) \wedge (\neg C \vee \neg D \vee \neg E) \wedge (\neg A \vee \neg C \vee E)$$

  - Goal: find if satisfactory assignment exists.
  - State: assignment to variables
  - ***f***(*s*): # satisfied clauses




Wiki

# Hill Climbing

One approach to such optimization problems.

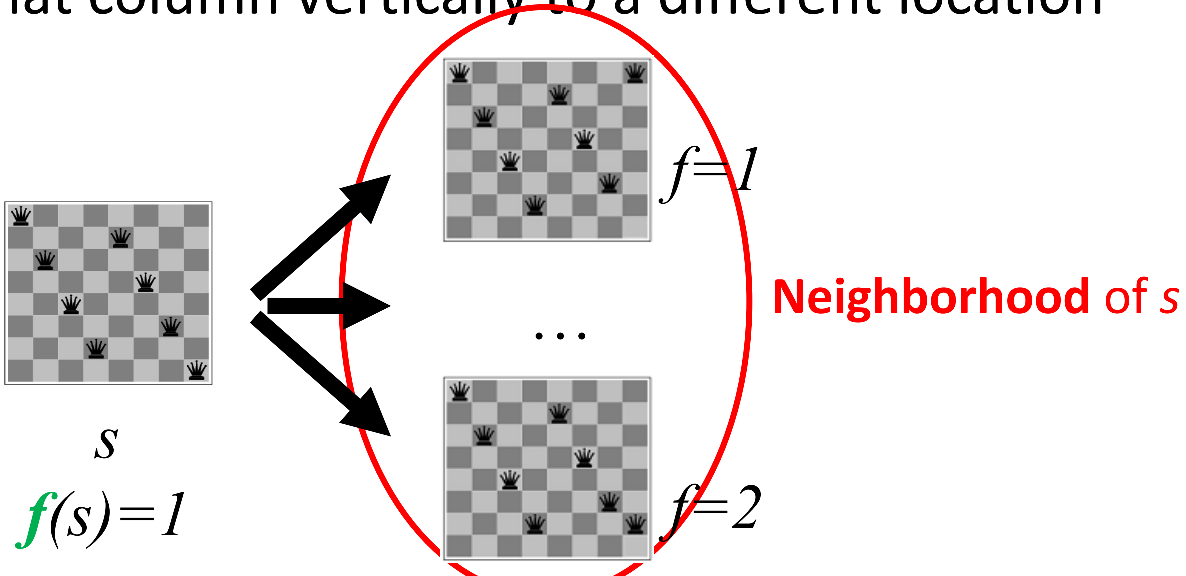
- Basic idea: move to a neighbor with a better ***f***(*s*)

- **Q**: how do we define **neighbor**?
  - Not as obvious as our successors in search
  - Problem-specific
  - As we'll see, needs a careful choice

# Defining Neighbors: n Queens

In n Queens, a simple possibility:

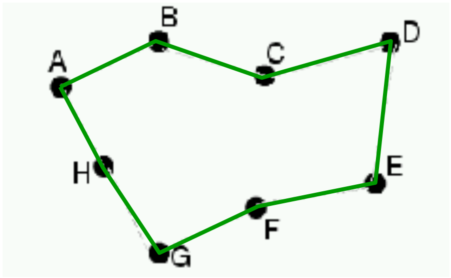
- Look at the **most-conflicting column** (ties? right-most one)
- Move queen in that column vertically to a different location



# Defining Neighbors: TSP

For TSP, can do something similar:

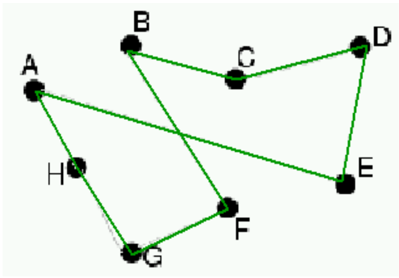
- Define neighbors by small changes
- Example: 2-change: A-E and B-F




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

flip

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



# Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Starting state: TFTTT

(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 Neighbors

**Q**: What's a **neighbor?**

- **Vague definition**. For a given problem structure, neighbors are states that can be produced by a small change
- **Tradeoff**!
  - Too small? Will get struck.
  - Too big? Not very efficient

- **Q**: how to pick a neighbor? Greedy
- **Q**: terminate? When no neighbor has bigger value

# Hill Climbing Algorithm

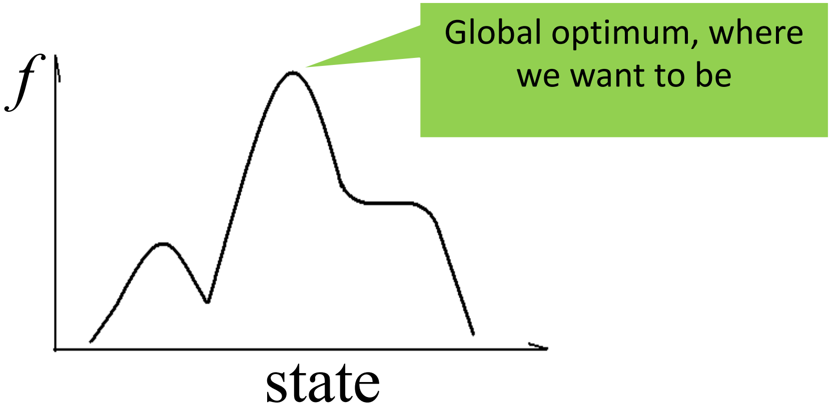
**Pseudocode:**

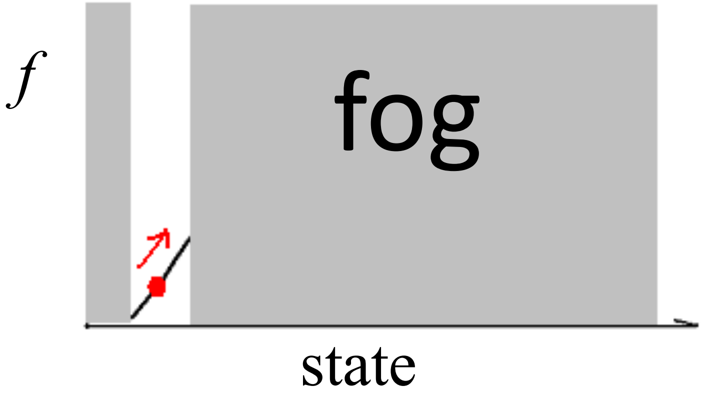
1. Pick initial state $s$
2. Pick $t$ in **neighbors**($s$) with the largest $\boldsymbol{f}(t)$
3. if $\boldsymbol{f}(t) \leq \boldsymbol{f}(s)$ THEN stop, return $s$
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

# Hill Climbing: Local Optima

**Q**: Why is it called hill climbing?



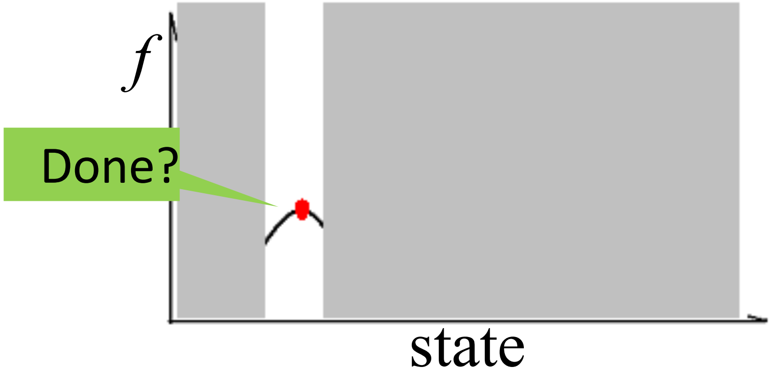



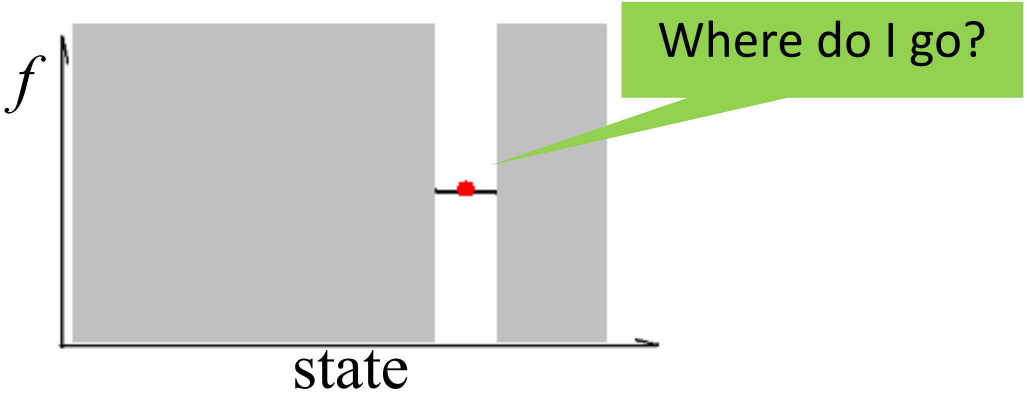

L: What’s actually going on.

R: What we get to see.

# Hill Climbing: Local Optima

Note the **local optima**. How do we handle them?



# Escaping Local Optima

**Simple idea 1**: random restarts

- Stuck: pick a random new starting point, re-run.
- Do *k* times, return best of the *k*.

**Simple idea 2**: reduce greed

- "Stochastic" hill climbing: randomly select between neighbors
- Probability proportional to the value of neighbors

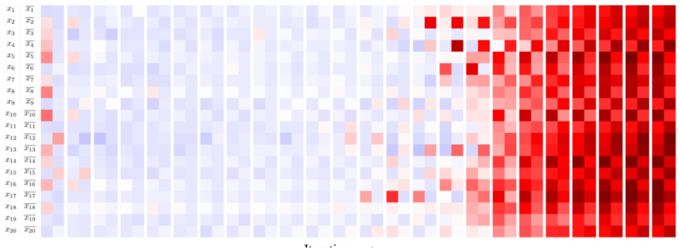
# Hill Climbing: Variations

**Q**: neighborhood too large?

- Generate random neighbors, **one at a time**. Take the better one.

**Q**: relax requirement to always go up?

- Often useful for harder problems
- 3SAT algorithm: Walk-SAT



D. Selsam

# Break & Quiz

**Q 1.1**: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i)
- B. (i), (ii)
- C. (i), (iii)
- D. All of the above

# Break & Quiz

**Q 1.1**: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i)
- B. (i), (ii)
- **C. (i), (iii)**
- D. All of the above

# Break & Quiz

**Q 1.1**: Hill climbing and SGD are related by

(i) Both head towards optima

(ii) Both require computing a gradient

(iii) Both will find the global optimum for a convex problem

- A. (i) (No: (iii) also true since convexity->local optima are global)
- B. (i), (ii) (No: (ii) is false. Hill-climbing looks at neighbors only.)
- **C. (i), (iii)**
- D. All of the above (No: (ii) false, as above.)

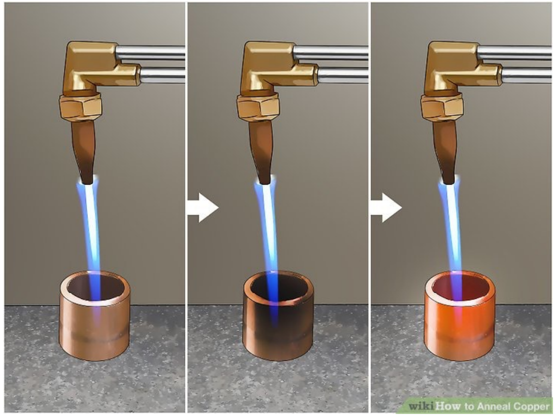
# Simulated Annealing

A more sophisticated optimization approach.

- **Idea**: allow some downhill moves at first, then be pickier over time
- Pseudocode:

Pick initial state s; T=1
For $k$ = 0 through $K$:
  $T \leftarrow T$*0.99 *(cool down)*
  Pick a random neighbour $t \leftarrow$ neighbor($s$)
  If $f(s) \leq f(t)$, then $s \leftarrow t$
  Else with prob. $P(f(s), f(t), \boldsymbol{T})$ still do $s \leftarrow t$
**Output**: the best state ever seen

The interesting bit




wikihow.com

# Simulated Annealing: Picking Probability

How do we pick probability P(f(s), f(t), Temp)?

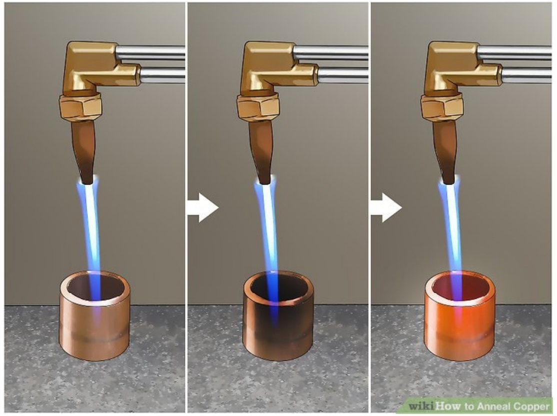
- Decrease with temperature
- Decrease with gap ***f***(*s*) - ***f***(*t*):

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

- Temperature cools over time.
  - So: high temperature, accept any *t*
  - But, low temperature, behaves like hill-climbing
  - Still, ***f***(*s*) - ***f***(*t*) plays a role: if big, replacement probability low.

# Simulated Annealing: Picking Parameters

- Have to balance the various parts., e.g., cooling schedule.
  - Too fast: becomes hill climbing, stuck in local optima
  - Too slow: takes too long.
- Combines with variations (e.g., with random restarts)
  - Probably should try hill-climbing first though.
- Inspired by cooling of metals
  - We'll see one more alg. inspired by nature




wikihow.com

# Break & Quiz

**Q 2.1**: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $Temp_{t+1} = Temp_t * 1.25$
B. $Temp_{t+1} = Temp_t$
C. $Temp_{t+1} = Temp_t * 0.8$
D. $Temp_{t+1} = Temp_t * 0.0001$

# Break & Quiz

**Q 2.1**: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $Temp_{t+1} = Temp_t * 1.25$
B. $Temp_{t+1} = Temp_t$
C. **$Temp_{t+1} = Temp_t * 0.8$**
D. $Temp_{t+1} = Temp_t * 0.0001$

# Break & Quiz

**Q 2.1**: Which of the following is likely to give the best cooling schedule for simulated annealing?

A. $Temp_{t+1} = Temp_t * 1.25$ (No, temperate is increasing)
B. $Temp_{t+1} = Temp_t$ (No, temperature is constant)
C. **$Temp_{t+1} = Temp_t * 0.8$**
D. $Temp_{t+1} = Temp_t * 0.0001$ (Cools too fast---basically hill climbing)

# Break & Quiz

**Q 2.2**: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

- A. (i)
- B. (ii)
- C. (i) and (ii)
- D. (ii) and (iii)

# Break & Quiz

**Q 2.2**: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a graph that involve all edges
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

- A. (i)
- B. (ii)
- **C. (i) and (ii)**
- D. (ii) and (iii)

# Break & Quiz

**Q 2.2**: Which of the following would be better to solve with simulated annealing than A* search?

i. Finding the smallest set of vertices in a complete graph (i.e., all nodes connected)
ii. Finding the fastest way to schedule jobs with varying runtimes on machines with varying processing power
iii. Finding the fastest way through a maze

- A. (i) (No, (ii) better: huge number of states, don't care about path)
- B. (ii) (No, (i) complete graph might have too many edges for A*)
- **C. (i) and (ii)**
- D. (ii) and (iii) (No, (iii) is good for A*: few successors, want path)

# Genetic Algorithms

Another optimization approach based on nature

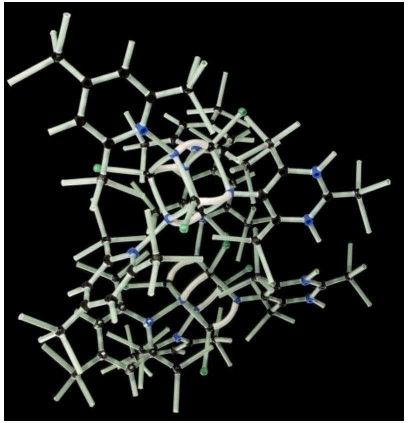
- Survival of the fittest!

# Evolution Review

Encode genetic information in DNA (four bases)

- A/C/T/G: nucleobases acting as symbols

- Two types of changes
  - Crossover: exchange between parents’ codes
  - Mutation: rarer random process
    - Happens at individual level



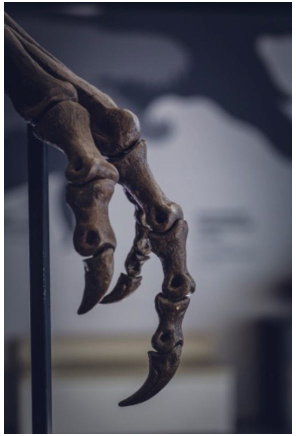
# Natural Selection

Competition for resources

- Organisms better fit ➔ better probability of reproducing
- Repeated process: fit become larger proportion of population

Goal: use these principles for optimization

- New terminology: state s ‘**individual**’
- Value $f(s)$ is now the ‘**fitness**’

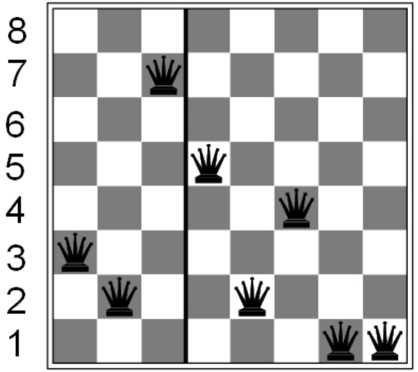


# Genetic Algorithms Setup I

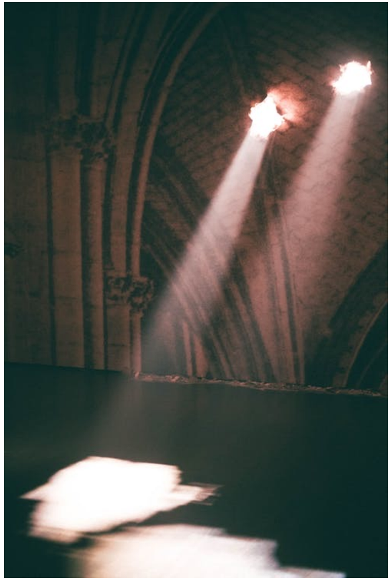
Keep around a fixed number of states/individuals

- A bit like beam search
- Call this the **population**

For our n Queens game example, an individual:



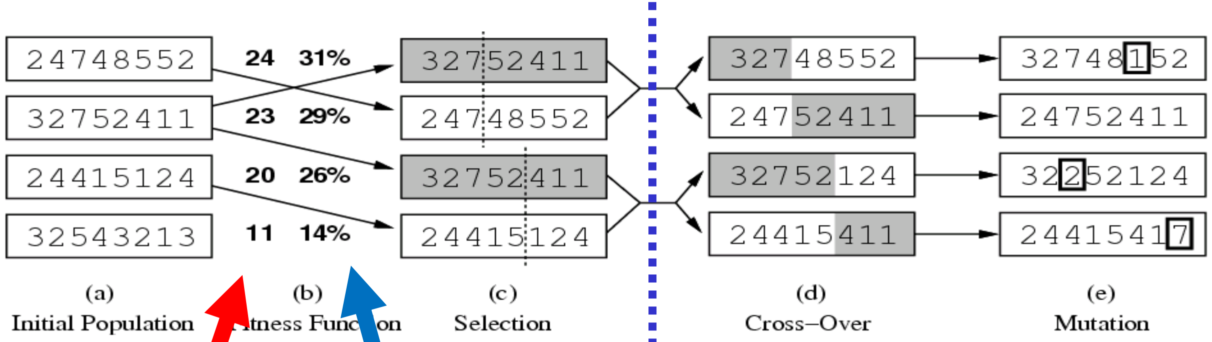

(3 2 7 5 2 4 1 1)



# Genetic Algorithms Setup II

Goal of genetic algorithms: optimize using principles inspired by mechanism for evolution

- E.g., analogous to **natural selection, cross-over**, and **mutation**




# of non-attacking pairs

prob. reproduction ∝ fitness

→ Next generation

# Genetic Algorithms Pseudocode

## Just one variant:

1. Let $s_1, \ldots, s_N$ be the current population
2. Let $p_i = \boldsymbol{f}(s_i) / \Sigma_j \boldsymbol{f}(s_j)$ be the reproduction probability
3. for $k = 1$; $k<N$; $k$+=2
   - parent1 = sample with replacement according to $p$
   - parent2 = sample with replacement according to $p$
   - 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

# Reproduction: Proportional Selection

Reproduction probability: $p_i = \boldsymbol{f}(s_i) \,/\, \Sigma_j \boldsymbol{f}(s_j)$

- **Example**: $\Sigma_j \boldsymbol{f}(s_j)$ = 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% |

# Example: Scheduling Courses

Let's run through an example:

- **5 courses: A,B,C,D,E**
- *3 time slots:* <u>M</u>on/Wed, <u>T</u>ue/Thu, <u>F</u>ri/Sat
- Students wish to enroll in three courses
- Goal: maximize student enrollment

| Courses | Students |
|---|---|
| A B C | 2 |
| A B D | 7 |
| A D E | 3 |
| B C D | 4 |
| B D E | 10 |
| C D E | 5 |

# Example: Scheduling Courses

Let’s run through an example:

- State: course assignment to time slot

| | | | | |
|---|---|---|---|---|
| M | M | F | T | M |
| A | B | C | D | E |

$= \mathtt{MMFTM}$

| Courses | Students |
|---|---|
| A B C | 2 |
| A B D | 7 |
| A D E | 3 |
| B C D | 4 |
| B D E | 10 |
| C D E | 5 |

- Here:
  – Courses A, B, E scheduled Mon/Wed
  – Course D scheduled Tue/Thu
  – Course C scheduled Fri/Sat

# Example: Scheduling Courses

Value of a state? Say `MMFTM`

| Courses | Students | Can enroll? |
|---|---|---|
| A B C | 2 | No |
| A B D | 7 | No |
| A D E | 3 | No |
| B C D | 4 | Yes |
| B D E | 10 | No |
| C D E | 5 | Yes |

- Here 4+5=9 students can enroll in desired courses

# Example: Scheduling Courses

## First step:

- Randomly initialize and evaluate states

| | |
|---|---|
| `MMFTM` = 9 | `MMFTM` = 26% |
| `TTFMM` = 4 | `TTFMM` = 11% |
| `FMTTF` = 19 | `FMTTF` = 54% |
| `MTTTF` = 3 | `MTTTF` = 9% |

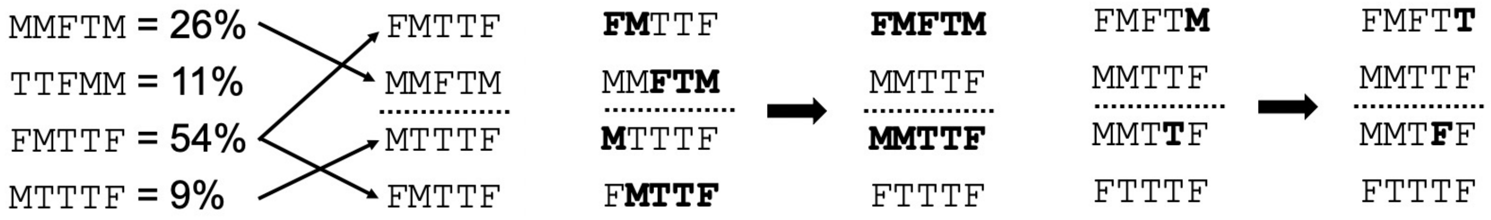
- Calculate reproduction probabilities

| Courses | Students |
|---|---|
| A B C | 2 |
| A B D | 7 |
| A D E | 3 |
| B C D | 4 |
| B D E | 10 |
| C D E | 5 |

# Example: Scheduling Courses

Next steps:

- Select parents using reproduction probabilities
- Perform crossover
- Randomly mutate new children



# Example: Scheduling Courses

Continue:

- Now, get our function values for updated population
- Calculate reproduction probabilities

| | |
|---|---|
| `FMFTT` = 11 | `FMFTT` = 39% |
| `MMTTF` = 13 | `MMTTF` = 46% |
| `MMTFF` = 4 | `MMTFF` = 14% |
| `FTTTF` = 0 | `FTTTF` = 0% |

| Courses | Students |
|---|---|
| A B C | 2 |
| A B D | 7 |
| A D E | 3 |
| B C D | 4 |
| B D E | 10 |
| C D E | 5 |

# Variations & Concerns

Many **possibilities**:

- Parents survive to next generation
- Use ranking instead of exact value of ***f***(*s*) for reproduction probabilities (reduce influence of extreme f values)

Some **challenges**

- State encoding
- Lack of diversity: converge too soon
- Must pick a lot of parameters

# Summary

- Challenging optimization problems
  - First, try hill climbing. Simplest solution
- Simulated annealing
  - More sophisticated approach; helps with local optima
- Genetic algorithms
  - Biology-inspired optimization routine