CS 540 Introduction to Artificial Intelligence

Search III: Advanced Search (aka Optimization)

University of Wisconsin-Madison

Spring 2023

Outline

Homeworks:

- Homework 10 due Thursday
- Course evaluation due Friday

Class roadmap:

Tuesday, May 2	Advanced Search
Thursday, May 4	Ethics and Review
Friday, May 12 5:05 - 7:05pm	Final Exam

Advanced Search Overview

Advanced Search Overview

Problem Setting

Advanced Search Overview

How is a search problem defined?

Problem Setting

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Hill Climbing

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Hill Climbing

Genetic Algorithms

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Hill Climbing

Genetic Algorithms

What is difference between two?

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Hill Climbing

Genetic Algorithms

What is difference between two?

Neighbors

Local vs. global optima

Advanced Search Overview

Problem Setting

How is a search problem defined?

How different from other search types?

Hill Climbing

Neighbors
Local vs. global optima

Genetic Algorithms

Fitness
Population
Cross-over
Mutation

What is difference between two?

Outline

Outline

- Advanced Search & Hill-climbing
  - More difficult problems, basics, local optima, variations

Outline

- Advanced Search & Hill-climbing
  - More difficult problems, basics, local optima, variations
- Hill Climbing
  - Basic algorithm, local optima

Outline

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

Search vs. Optimization

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

Search vs. Optimization

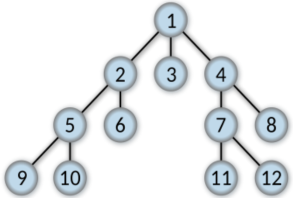
Before: wanted a **path** from start state to goal state

- Uninformed search, informed search

Search vs. Optimization

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

- Uninformed search, informed search

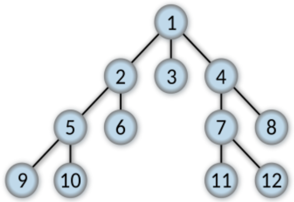


Wiki

Search vs. Optimization

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

- Uninformed search, informed search



Wiki

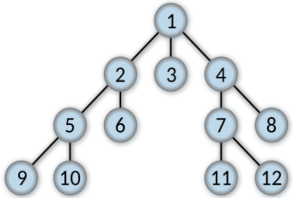
New setting: optimization

Search vs. Optimization

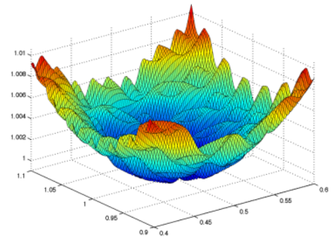
Before: wanted a **path** from start state to goal state

- Uninformed search, informed search

New setting: optimization



Wiki

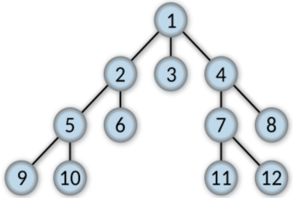


TuringFin

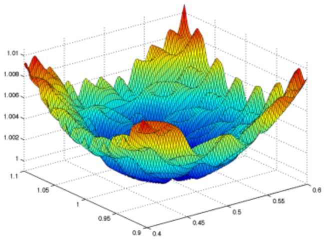
Search vs. Optimization

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

- Uninformed search, informed search



Wiki



TuringFin

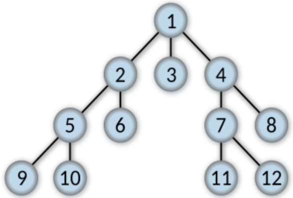
New setting: optimization

- States *s* have values ***f***(*s*)

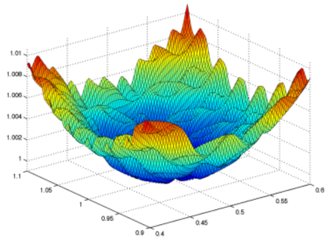
Search vs. Optimization

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

- Uninformed search, informed search



Wiki



TuringFin

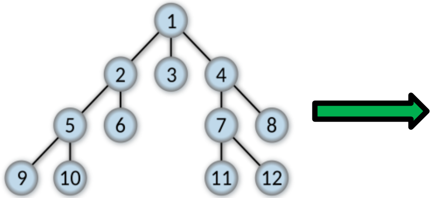
New setting: optimization

- States *s* have values ***f***(*s*)
- Want: Find *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)

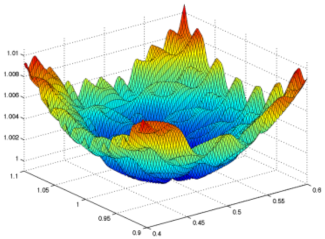
Search vs. Optimization

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: Find *s* with optimal value ***f***(*s*) (i.e, **optimize** over states)
- Challenging settings: **too many states** for previous search approaches, but maybe not a differentiable function for gradient descent.

Examples: n Queens

A classic puzzle:

Examples: n Queens

A classic puzzle:

- Place 8 queens on 8 x 8 chessboard so that no two have same row, column, or diagonal.

Examples: n Queens

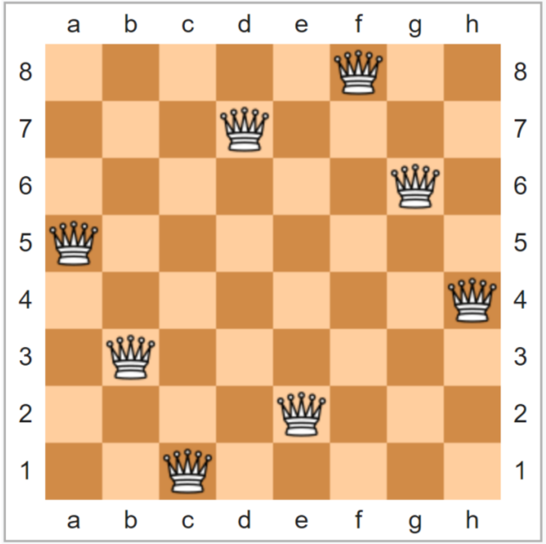
A classic puzzle:

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

Examples: n Queens

A classic puzzle:

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

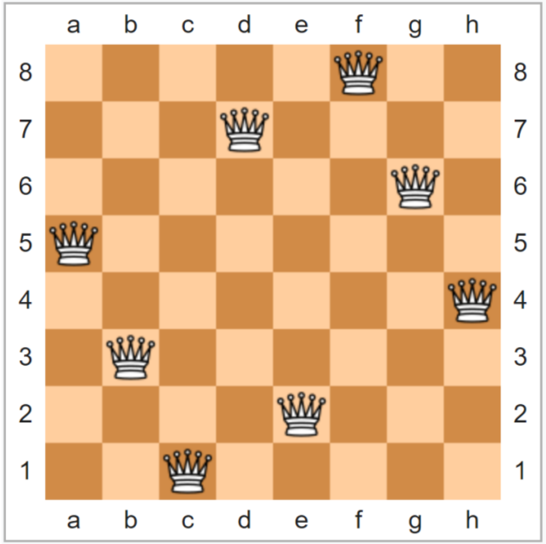


Wiki

Examples: n Queens

A classic puzzle:

- Place 8 queens on 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*)?



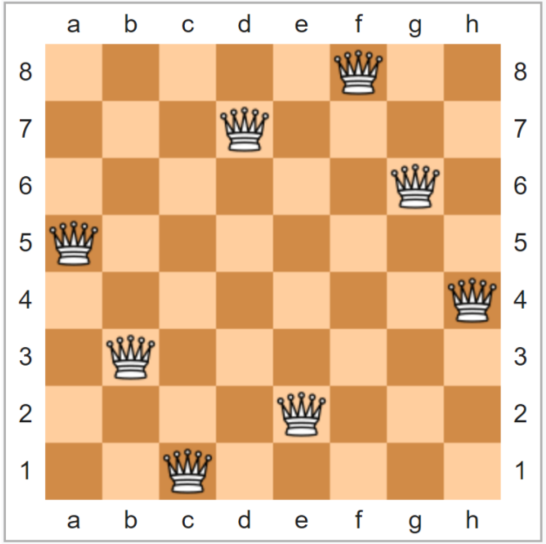
Wiki

Examples: n Queens

A classic puzzle:

- Place 8 queens on 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



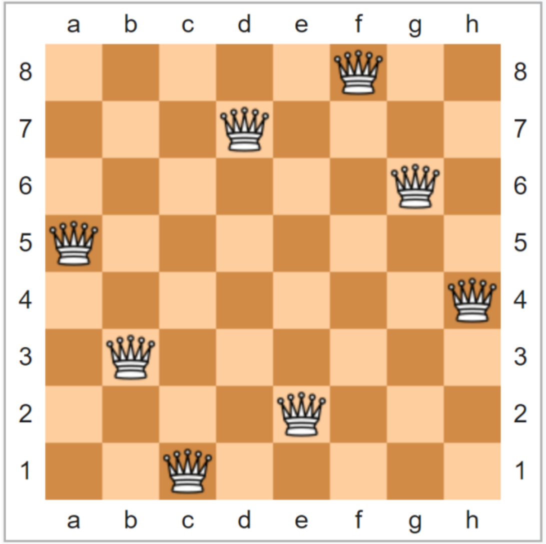
Wiki

Examples: n Queens

A classic puzzle:

- Place 8 queens on 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 non-conflicting queens



Wiki

Examples: TSP

Famous graph theory problem.

Examples: TSP

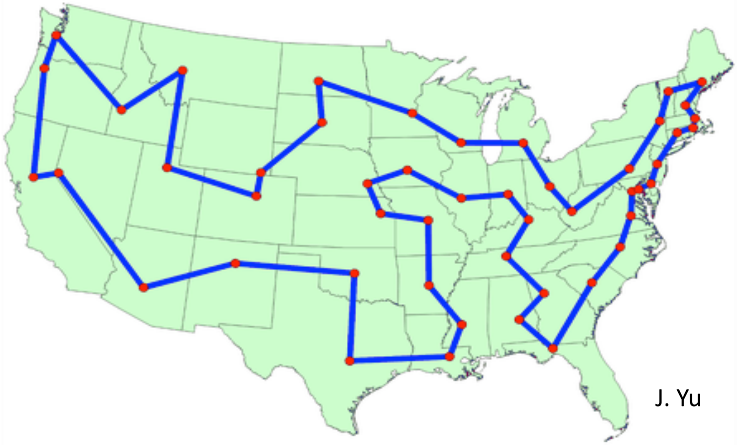
Famous graph theory problem.

- Get a graph G = (V,E). **Goal**: a path that visits each node exactly once and returns to the initial node (a **tour**).

Examples: TSP

Famous graph theory problem.

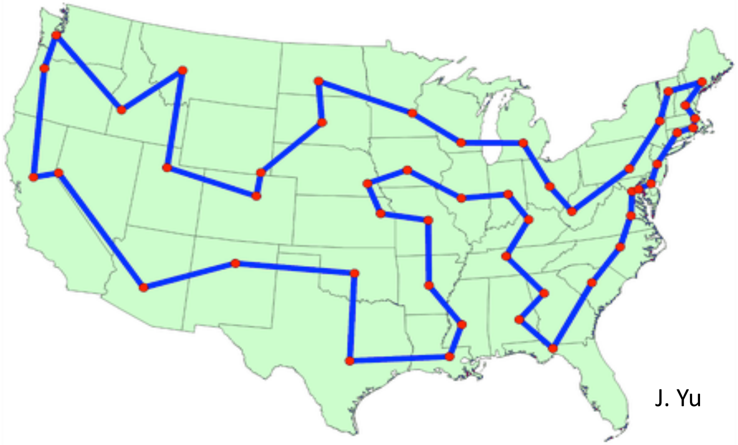
- Get a graph G = (V,E). **Goal**: a path that visits each node exactly once and returns to the initial node (a **tour**).



Examples: TSP

Famous graph theory problem.

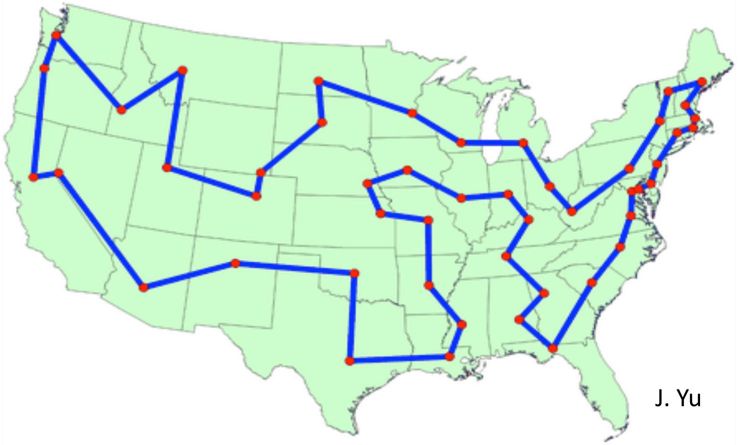
- Get a graph G = (V,E). **Goal**: a path that visits each node exactly once and returns to the initial node (a **tour**).
  - State: a particular tour (i.e., ordered list of nodes)



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



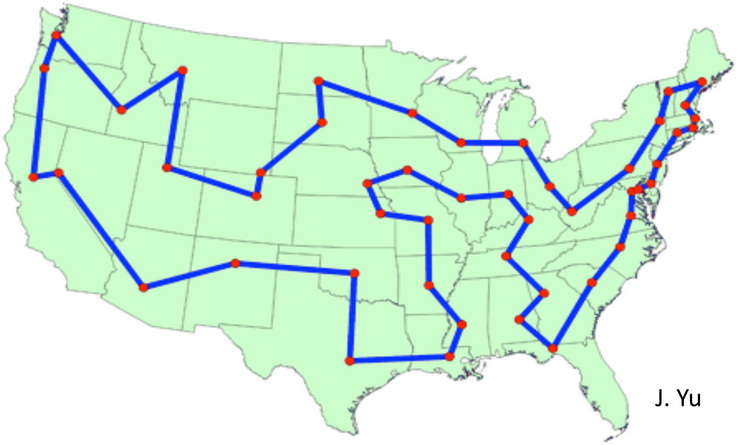
J. Yu

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)



Examples: Satisfiability

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

Examples: Satisfiability

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

- Recall our logic lecture. Conjunctive normal form

Examples: Satisfiability

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

- Recall our logic lecture. Conjunctive normal form

$$(A \lor \neg B \lor C) \land (\neg A \lor C \lor D) \land (B \lor D \lor \neg E) \land (\neg C \lor \neg D \lor \neg E) \land (\neg A \lor \neg C \lor E)$$

Examples: Satisfiability

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

- Recall our logic lecture. Conjunctive normal form

$$(A \lor \neg B \lor C) \land (\neg A \lor C \lor D) \land (B \lor D \lor \neg E) \land (\neg C \lor \neg D \lor \neg E) \land (\neg A \lor \neg C \lor E)$$

  - Goal: find if satisfactory assignment exists.

Examples: Satisfiability

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

- Recall our logic lecture. Conjunctive normal form

$$(A \lor \neg B \lor C) \land (\neg A \lor C \lor D) \land (B \lor D \lor \neg E) \land (\neg C \lor \neg D \lor \neg E) \land (\neg A \lor \neg C \lor E)$$

  - Goal: find if satisfactory assignment exists.
  - State: assignment to variables

Examples: Satisfiability

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

- Recall our logic lecture. Conjunctive normal form

$$(A \lor \neg B \lor C) \land (\neg A \lor C \lor D) \land (B \lor D \lor \neg E) \land (\neg C \lor \neg D \lor \neg E) \land (\neg A \lor \neg C \lor E)$$

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

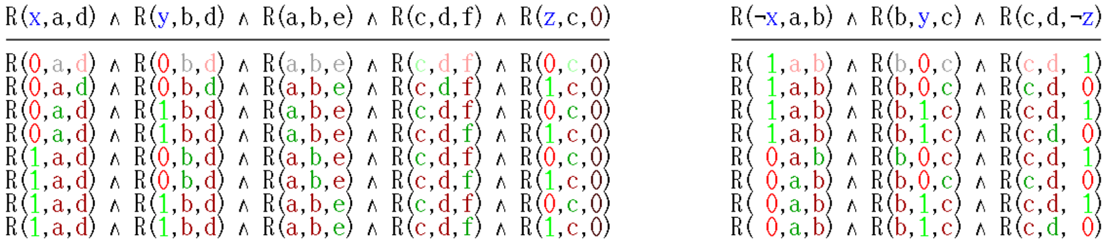
Examples: Satisfiability

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

Hill Climbing

One approach to such optimization problems

- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

Hill Climbing

One approach to such optimization problems

- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

Hill Climbing

One approach to such optimization problems

- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

- **Q**: how do we define **neighbor**?

Hill Climbing

One approach to such optimization problems

- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

- **Q**: how do we define **neighbor**?
  - Not as obvious as our successors in search

Hill Climbing

One approach to such optimization problems

- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

- **Q**: how do we define **neighbor**?
  - Not as obvious as our successors in search
  - Problem-specific

Hill Climbing

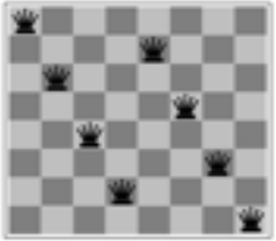
One approach to such optimization problems

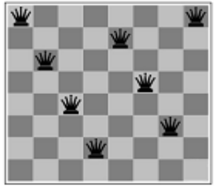
- Basic idea: start at one state, move to a neighbor with a better ***f***(*s*) value, repeat until no neighbors have better ***f***(*s*) value.

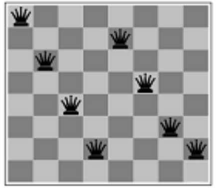
- **Q**: how do we define **neighbor**?
  - Not as obvious as our successors in search
  - Problem-specific
  - As we'll see, needs a careful choice

Defining Neighbors: n Queens

In n Queens, a simple possibility:



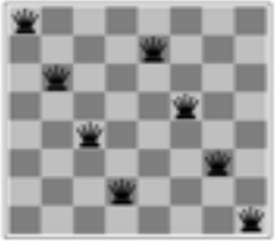


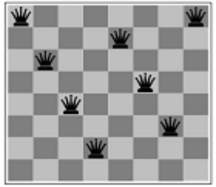


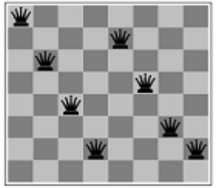
Defining Neighbors: n Queens

In n Queens, a simple possibility:

- Look at the **most-conflicting column** (ties? right-most one)



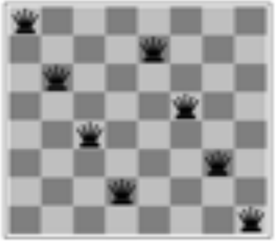


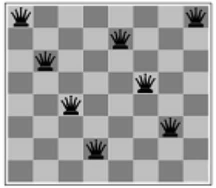


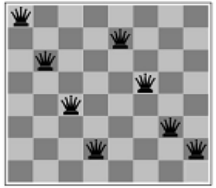
Defining Neighbors: n Queens

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



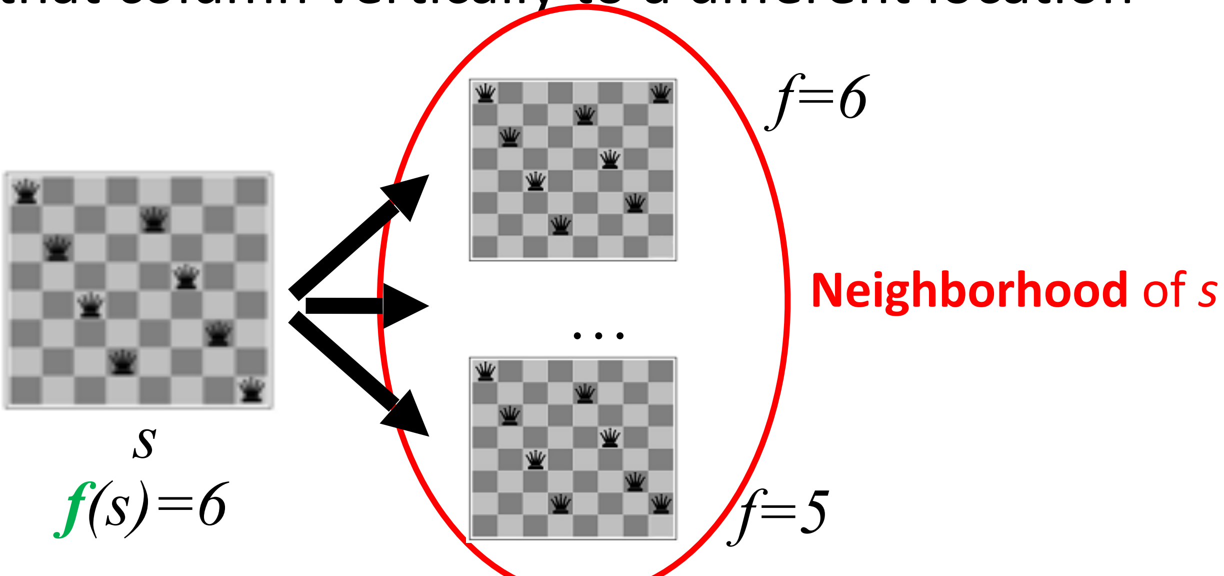




Defining Neighbors: n Queens

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:

Defining Neighbors: TSP

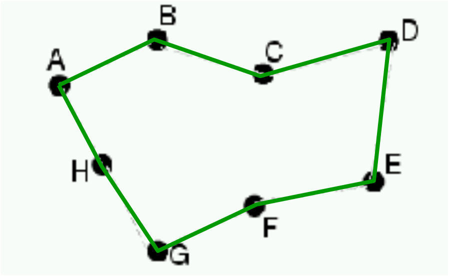
For TSP, can do something similar:

- Define neighbors by small changes

Defining Neighbors: TSP

For TSP, can do something similar:

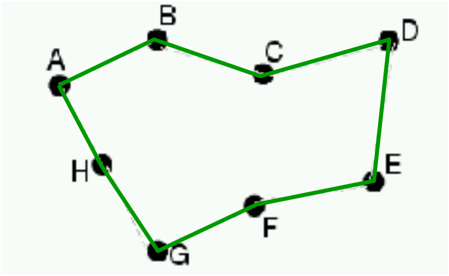
- Define neighbors by small changes



Defining Neighbors: TSP

For TSP, can do something similar:

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

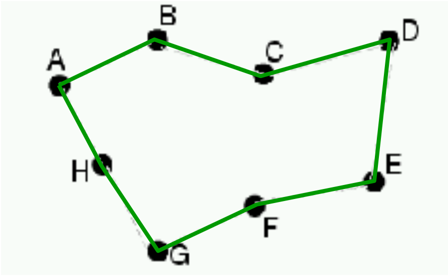


Defining Neighbors: TSP

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

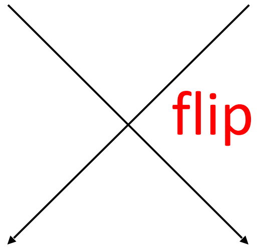


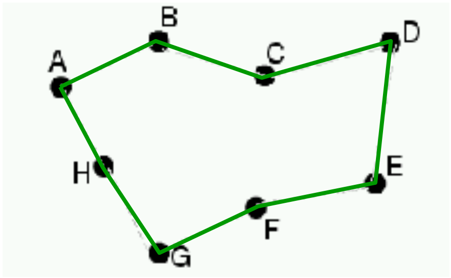
Defining Neighbors: TSP

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

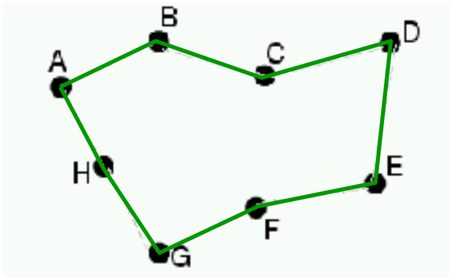




Defining Neighbors: TSP

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,

Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Starting state: (A=T, B=F, C=T, D=T, F=T)

Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Starting state: (A=T, B=F, C=T, D=T, F=T)

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

Defining Neighbors: SAT

For Boolean satisfiability,

- Define neighbors by flipping one assignment of one variable

Starting state: (A=T, B=F, C=T, D=T, F=T)

(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?**

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

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

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**!

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**!
  - Neighborhood too small? Will get struck.

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**!
  - Neighborhood too small? Will get struck.
  - Neighborhood too big? Not very efficient

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**!
  - Neighborhood too small? Will get struck.
  - Neighborhood too big? Not very efficient

- **Q**: how to pick a neighbor? Greedy

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**!
  - Neighborhood too small? Will get struck.
  - Neighborhood too big? Not very efficient

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

Hill Climbing Algorithm

Hill Climbing Algorithm

Pseudocode:

Hill Climbing Algorithm

Pseudocode:

1. Pick initial state *s*
2. Pick *t* in **neighbors**(*s*) with the best ***f***(*t*)
3. if ***f***(*t*) is not better than ***f***(*s*) THEN stop, return *s*
4. *s* ← *t*. goto 2.

Hill Climbing Algorithm

Pseudocode:

1. Pick initial state s
2. Pick t in **neighbors**(s) with the best $\boldsymbol{f}(t)$
3. if $\boldsymbol{f}(t)$ is not better than $\boldsymbol{f}(s)$ THEN stop, return s
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

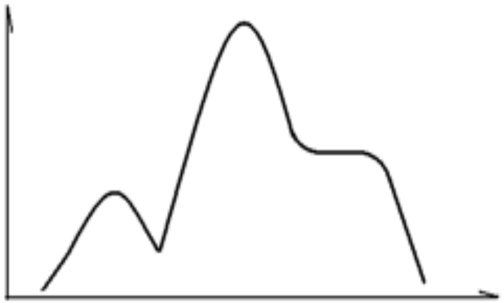
Hill Climbing Algorithm

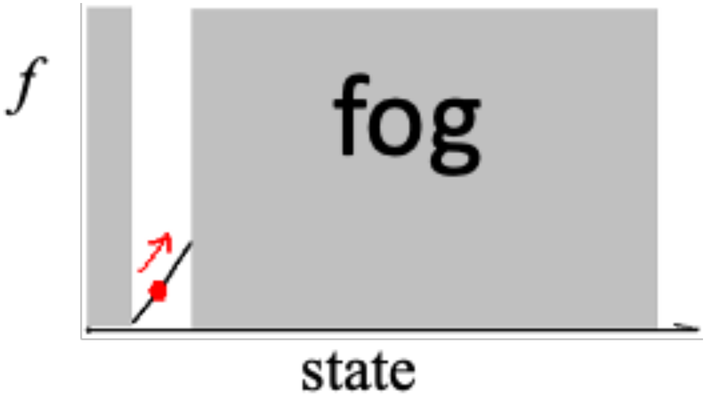
Pseudocode:

1. Pick initial state *s*
2. Pick *t* in **neighbors**(*s*) with the best ***f***(*t*)
3. if ***f***(*t*) is not better than ***f***(*s*) THEN stop, return *s*
4. $s \leftarrow t$. goto 2.

What could happen? **Local optima!**

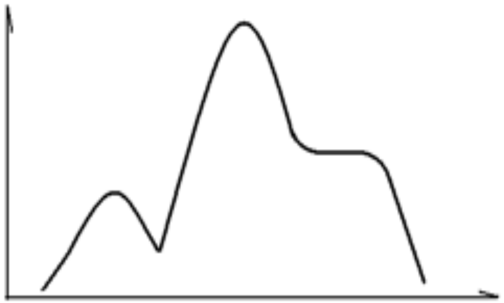
Hill Climbing: Local Optima

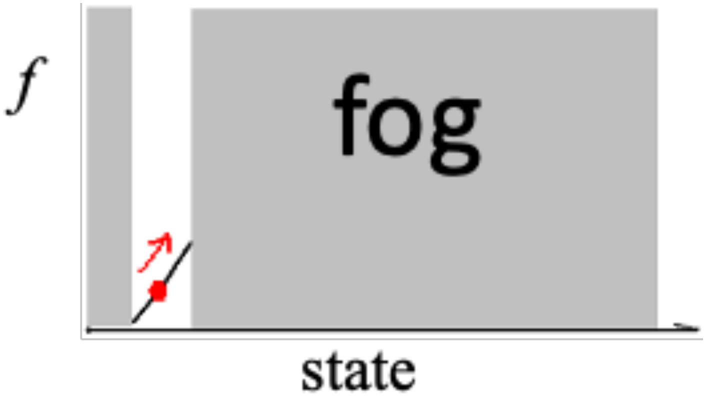




Hill Climbing: Local Optima

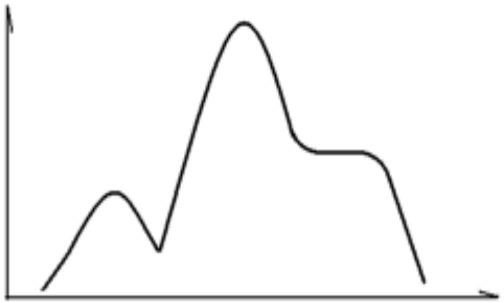
Q: Why is it called hill climbing?

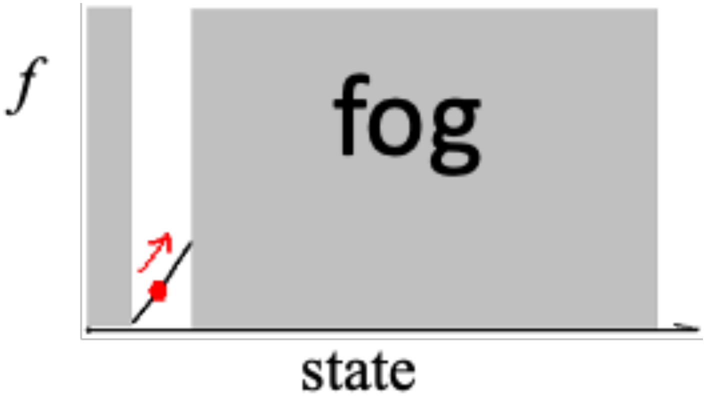




Hill Climbing: Local Optima

Q: Why is it called hill climbing?



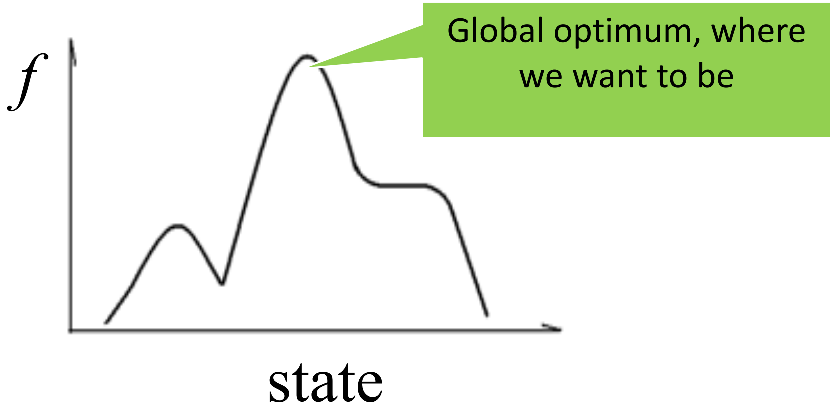


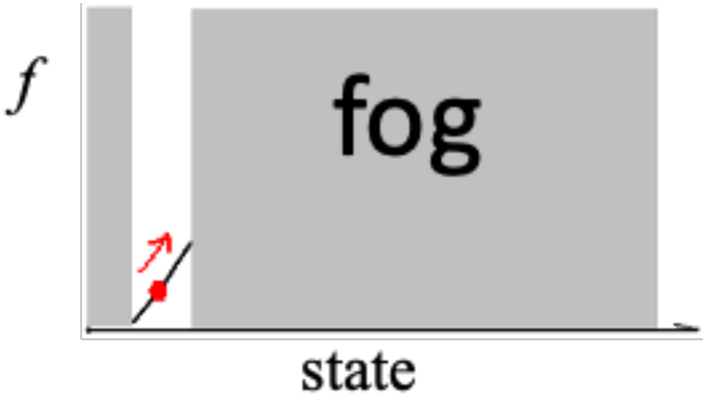
L: What's actually going on.

R: What we get to see.

Hill Climbing: Local Optima

Q: Why is it called hill climbing?



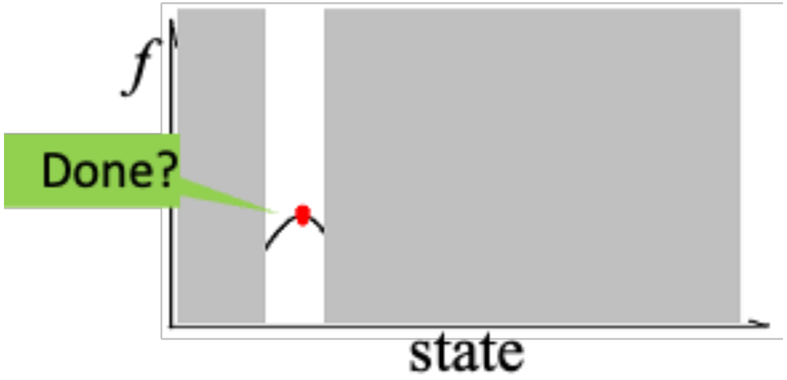


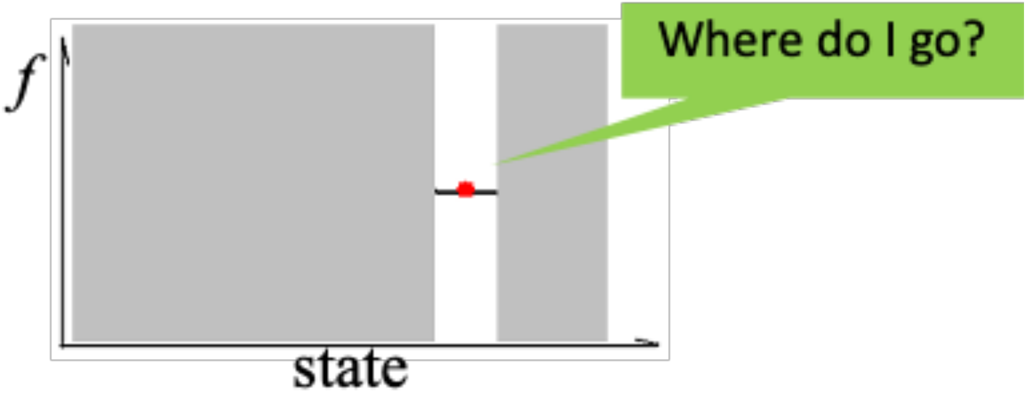
L: What's actually going on.

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

Escaping Local Optima

Simple idea 1: random restarts

- Stuck: pick a random new starting point, re-run.

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* runs.

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* runs.

Simple idea 2: reduce greed

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* runs.

Simple idea 2: reduce greed

- “Stochastic” hill climbing: randomly select between neighbors.

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* runs.

Simple idea 2: reduce greed

- "Stochastic" hill climbing: randomly select between neighbors.
- Probability of selecting a neighbor should be proportional to the value of that neighbor.

Hill Climbing: Variations

Q: neighborhood too large?

Hill Climbing: Variations

Q: neighborhood too large?

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

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?

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

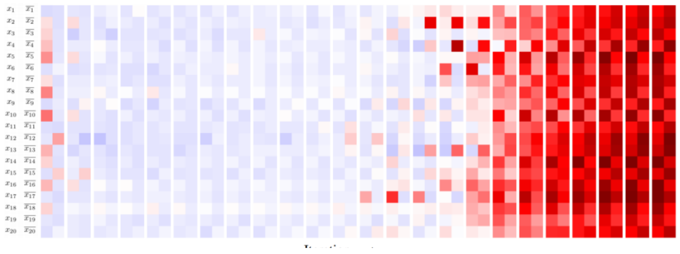
Hill Climbing: Variations

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



D. Selsam

Break & Quiz

Q 1.1: Hill climbing and stochastic gradient descent 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 (problem where all optima have the same value).

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

Break & Quiz

Q 1.1: Hill climbing and stochastic gradient descent 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 (problem where all optima have the same value).

- A. (i)
- B. (i), (ii) (No: (ii) is false. Hill-climbing looks at neighbors only.)
- **C. (i), (iii)**
- D. All of the above

Break & Quiz

Q 2.2: Which of the following would be better to solve with hill climbing rather 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 hill climbing rather 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 hill climbing rather 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) (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

Genetic Algorithms

Optimization approach based on nature

- Survival of the fittest!

Genetic Algorithms

Optimization approach based on nature

- Survival of the fittest!

Evolution Review

Encode genetic information in DNA (four bases)

Evolution Review

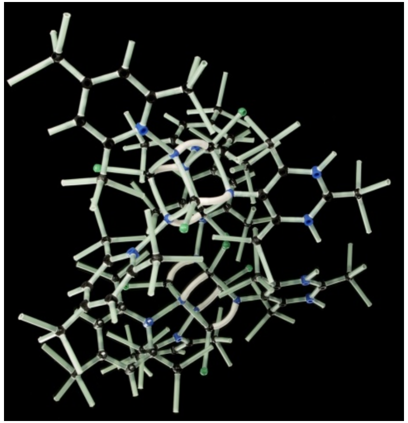
Encode genetic information in DNA (four bases)

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

Evolution Review

Encode genetic information in DNA (four bases)

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

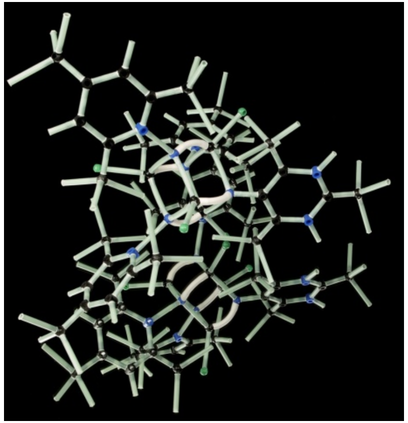


Evolution Review

Encode genetic information in DNA (four bases)

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

- Two types of changes

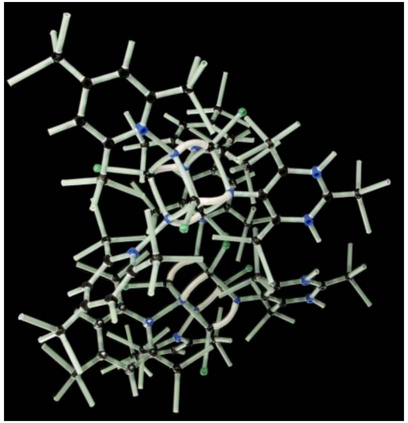


Evolution Review

Encode genetic information in DNA (four bases)

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

- Two types of changes
  - Crossover: exchange between parents' codes

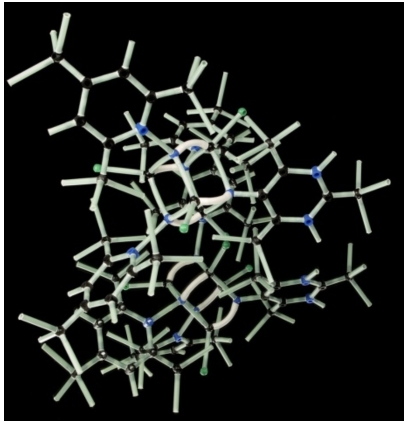


Evolution Review

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

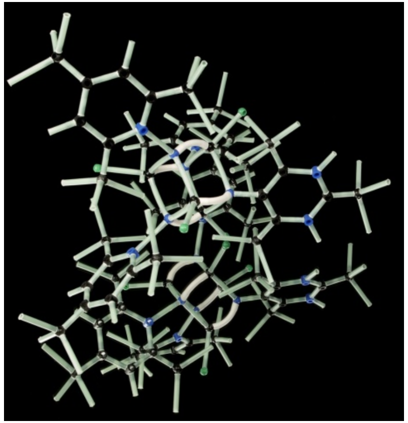


Evolution Review

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

Natural Selection

Competition for resources

- Organisms with better fitness ➔ better probability of reproducing

Natural Selection

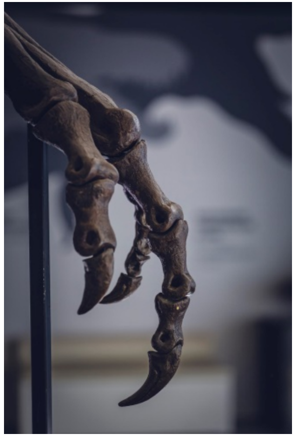
Competition for resources

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

Natural Selection

Competition for resources

- Organisms with better fitness → better probability of reproducing
- Repeated process: fit become larger proportion of population

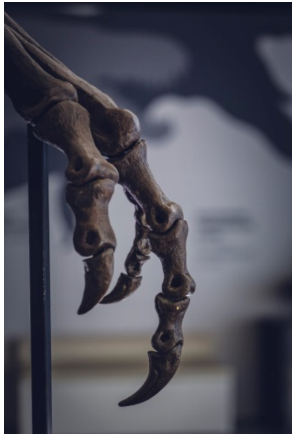


Natural Selection

Competition for resources

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

Goal: use these principles for optimization



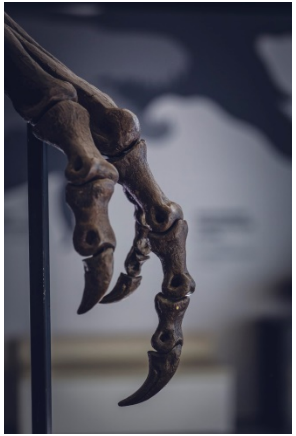
Natural Selection

Competition for resources

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

Goal: use these principles for optimization

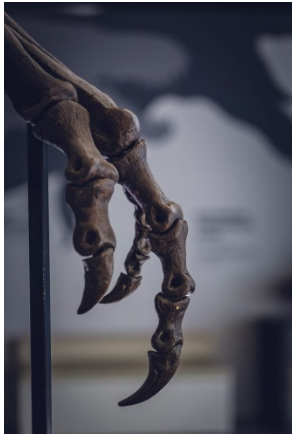
– New terminology: state is '**individual**'



Natural Selection

Competition for resources

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

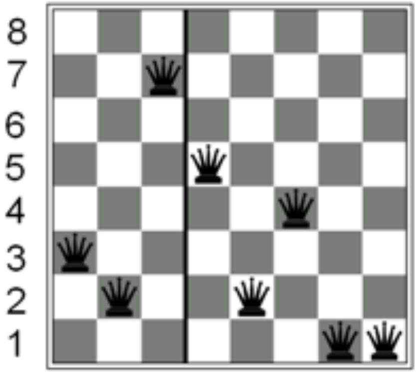


Goal: use these principles for optimization

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

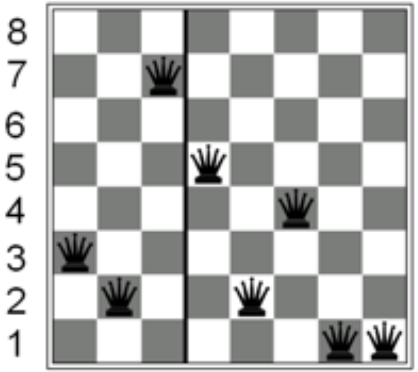
Genetic Algorithms Setup I

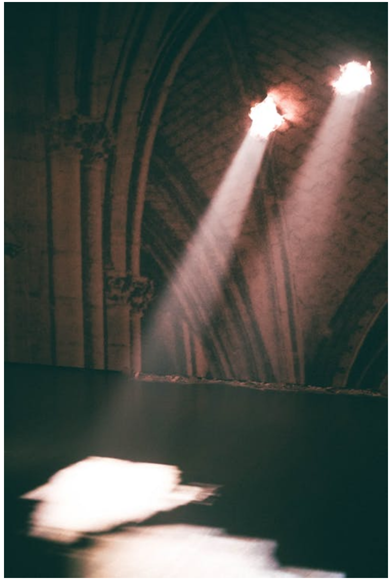
Keep around a fixed number of states/individuals



Genetic Algorithms Setup I

Keep around a fixed number of states/individuals

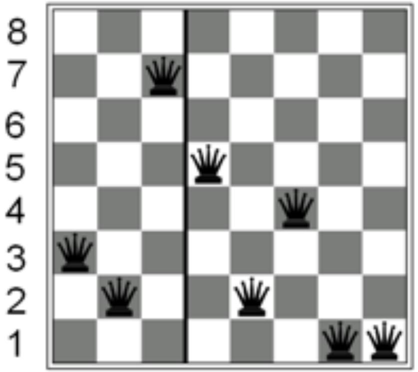


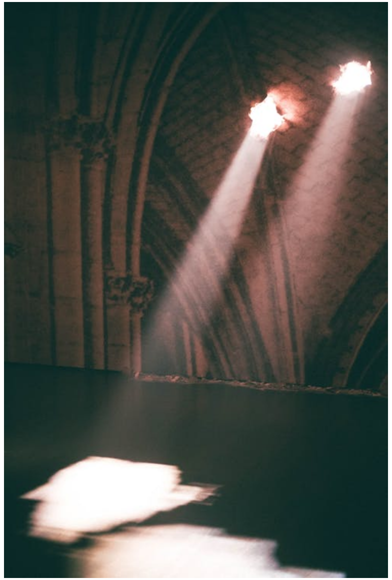


Genetic Algorithms Setup I

Keep around a fixed number of states/individuals

- Call this the **population**



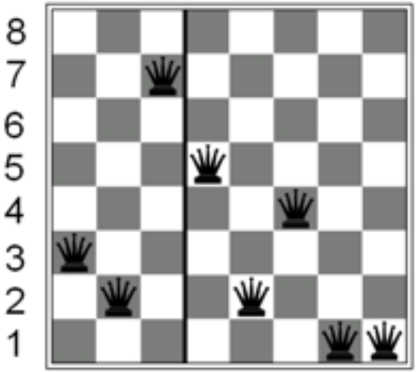


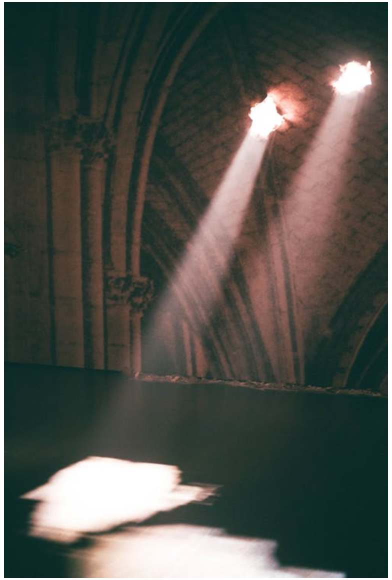
Genetic Algorithms Setup I

Keep around a fixed number of states/individuals

- Call this the **population**

For our n Queens game example, an individual:



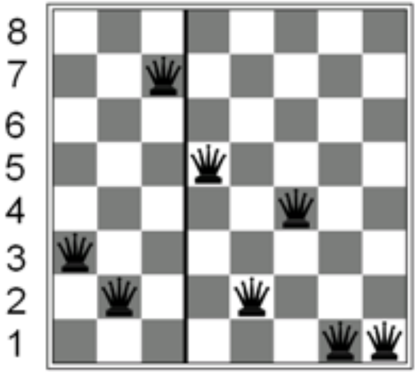


Genetic Algorithms Setup I

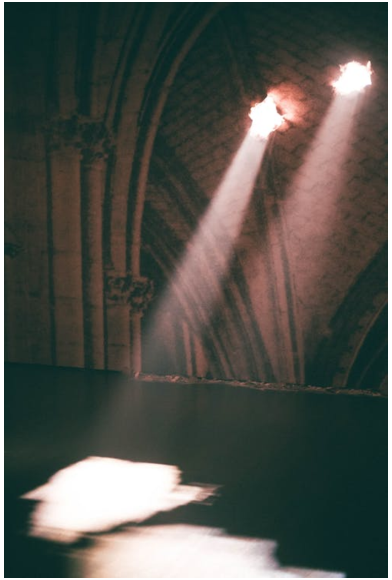
Keep around a fixed number of states/individuals

- Call this the **population**

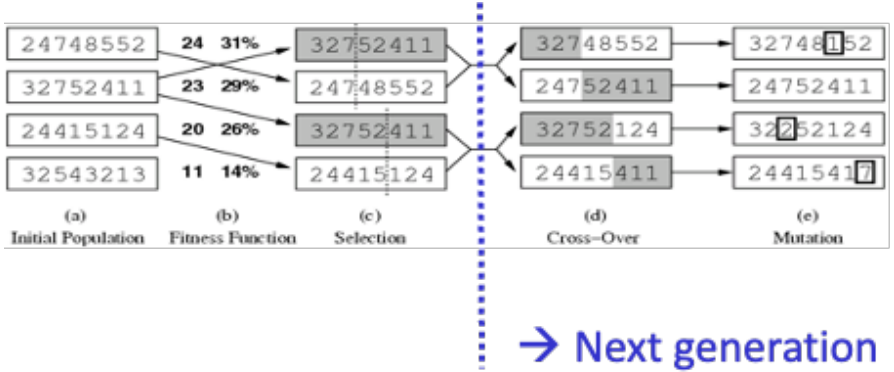
For our n Queens game example, an individual:



(3 2 7 5 2 4 1 1)



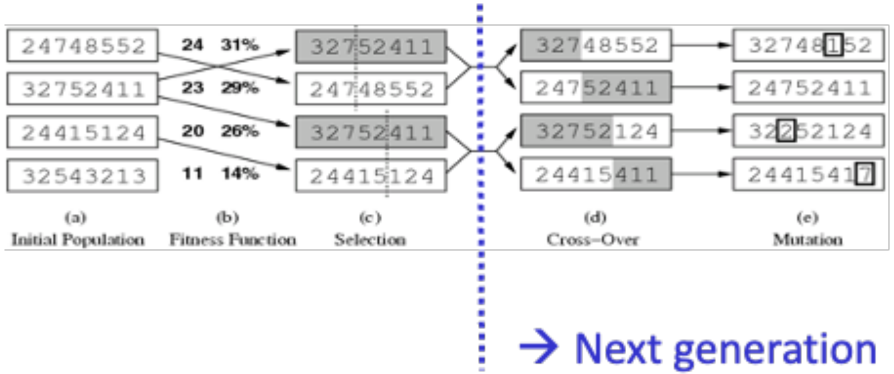
Genetic Algorithms Setup II



Genetic Algorithms Setup II

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

- Analogous to **natural selection, cross-over**, and **mutation**



Genetic Algorithms Setup II

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

- Analogous to **natural selection, cross-over**, and **mutation**

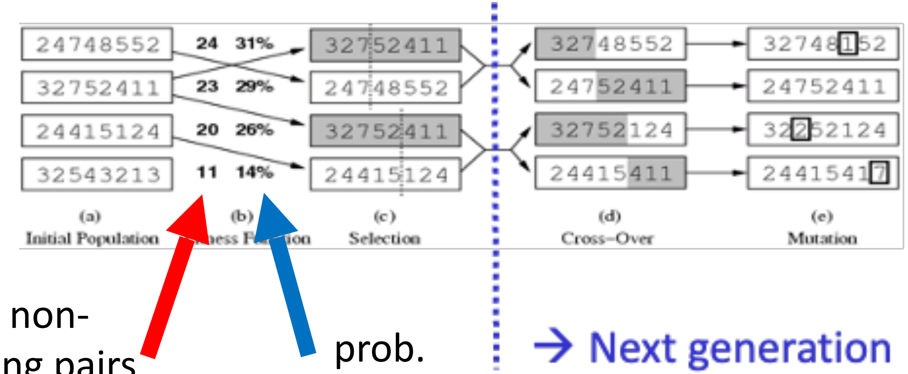
# of non-attacking pairs

→ Next generation

Genetic Algorithms Setup II

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

- 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, …, 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 }←{ t }. Repeat

Reproduction: Proportional Selection

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

Reproduction: Proportional Selection

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

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

Reproduction: Proportional Selection

Reproduction probability: $p_i = f(s_i) / \Sigma_j f(s_j)$

- **Example**: $\Sigma_j f(s_j) = 5+20+11+8+6=50$

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

Reproduction: Proportional Selection

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

- **Example**: $\Sigma_j \boldsymbol{f}(s_j) = 5+20+11+8+6=50$

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

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:

Example: Scheduling Courses

Let’s run through an example:

- **5 courses: A,B,C,D,E**

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

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

Example: Scheduling Courses

Let’s run through an example:

- **5 courses: A,B,C,D,E**
- *3 time slots:* Mon/Wed, Tue/Thu, Fri/Sat
- Students wish to enroll in three courses

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:

- **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:

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

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

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

= 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

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

$= \mathrm{MMFTM}$

- Here:

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

$= \mathrm{MMFTM}$

- Here:
  - Courses A, B, E scheduled Mon/Wed

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

$= \mathtt{MMFTM}$

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

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

$= \mathtt{MMFTM}$

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

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

Example: Scheduling Courses

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

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:

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

First step:

- Randomly initialize and evaluate states

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

First step:

- Randomly initialize and evaluate states

`MMFTM` = 9

`TTFMM` = 4

`FMTTF` = 19

`MTTTF` = 3

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

First step:

- Randomly initialize and evaluate states

  `MMFTM` = 9

  `TTFMM` = 4

  `FMTTF` = 19

  `MTTTF` = 3

- 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

First step:

- Randomly initialize and evaluate states

MMFTM = 9

TTFMM = 4

FMTTF = 19

MTTTF = 3

MMFTM = 26%

TTFMM = 11%

FMTTF = 54%

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:

Example: Scheduling Courses

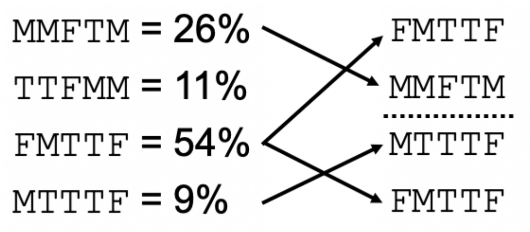
Next steps:

- Select parents using reproduction probabilities

Example: Scheduling Courses

Next steps:

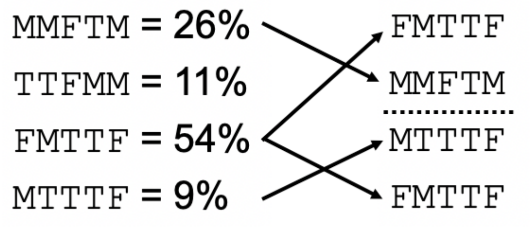
- Select parents using reproduction probabilities



Example: Scheduling Courses

Next steps:

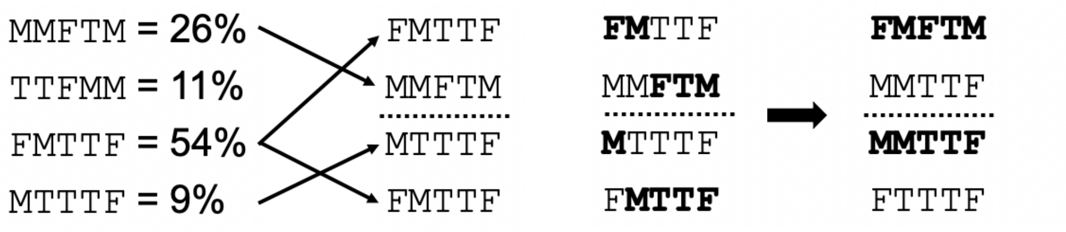
- Select parents using reproduction probabilities
- Perform crossover



Example: Scheduling Courses

Next steps:

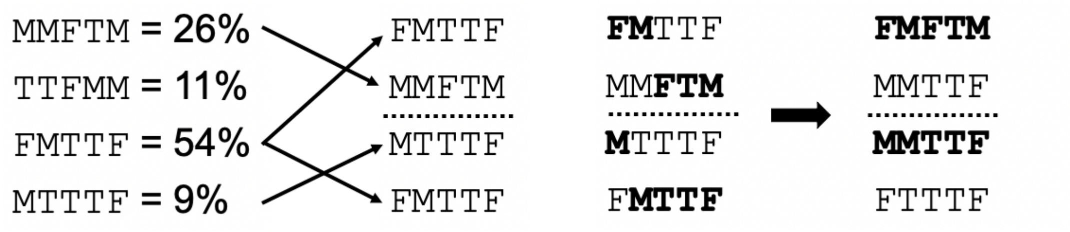
- Select parents using reproduction probabilities
- Perform crossover



Example: Scheduling Courses

Next steps:

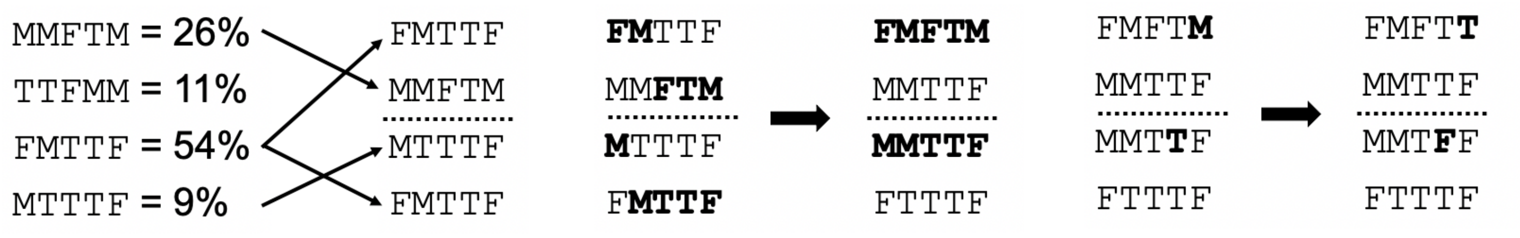
- Select parents using reproduction probabilities
- Perform crossover
- Randomly mutate new children



Example: Scheduling Courses

Next steps:

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



Example: Scheduling Courses

Continue:

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

Continue:

- Now, get our function values for updated population

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

Continue:

- Now, get our function values for updated population

`FMFTT` = 11

`MMTTF` = 13

`MMTFF` = 4

`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

Example: Scheduling Courses

Continue:

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

`FMFTT` = 11

`MMTTF` = 13

`MMTFF` = 4

`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

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**:

Variations & Concerns

Many **possibilities**:

- Parents survive to next generation

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)

Variations & Concerns

Many **possibilities**:

- Parents survive to next generation
- Use ranking instead of exact value of $\boldsymbol{f}(s)$ for reproduction probabilities (reduce influence of extreme f values)

Some **challenges**

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**

- Formulating a good state encoding

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**

- Formulating a good state encoding
- Lack of diversity: converge too soon

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**

- Formulating a good state encoding
- Lack of diversity: converge too soon
- Must pick a lot of parameters

Summary

- Challenging optimization problems
  - First, try hill climbing. Simplest solution
- Genetic algorithms
  - Biology-inspired optimization routine