Advanced Search
Hill climbing

Yingyu Liang

yliang@cs.wisc.edu

Computer Sciences Department

University of Wisconsin, Madison

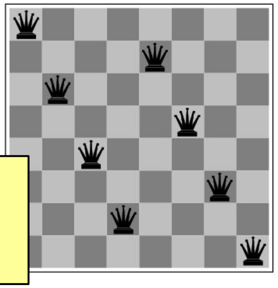
[Based on slides from Jerry Zhu, Andrew Moore http://www.cs.cmu.edu/~awm/tutorials]

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

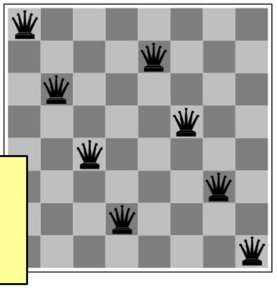
- N-queen: *f*(*s*) = number of conflicting queens in state *s*



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

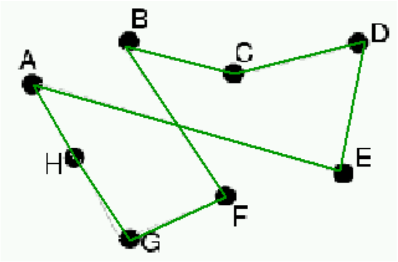
Examples

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



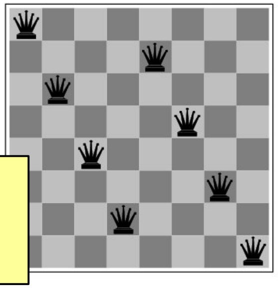
Note we want s with the lowest score $f(s)=0$. The techniques are the same. Low or high should be obvious from context.

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



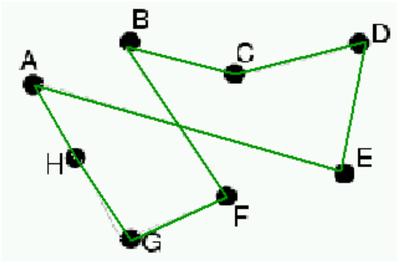
Examples

- 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

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

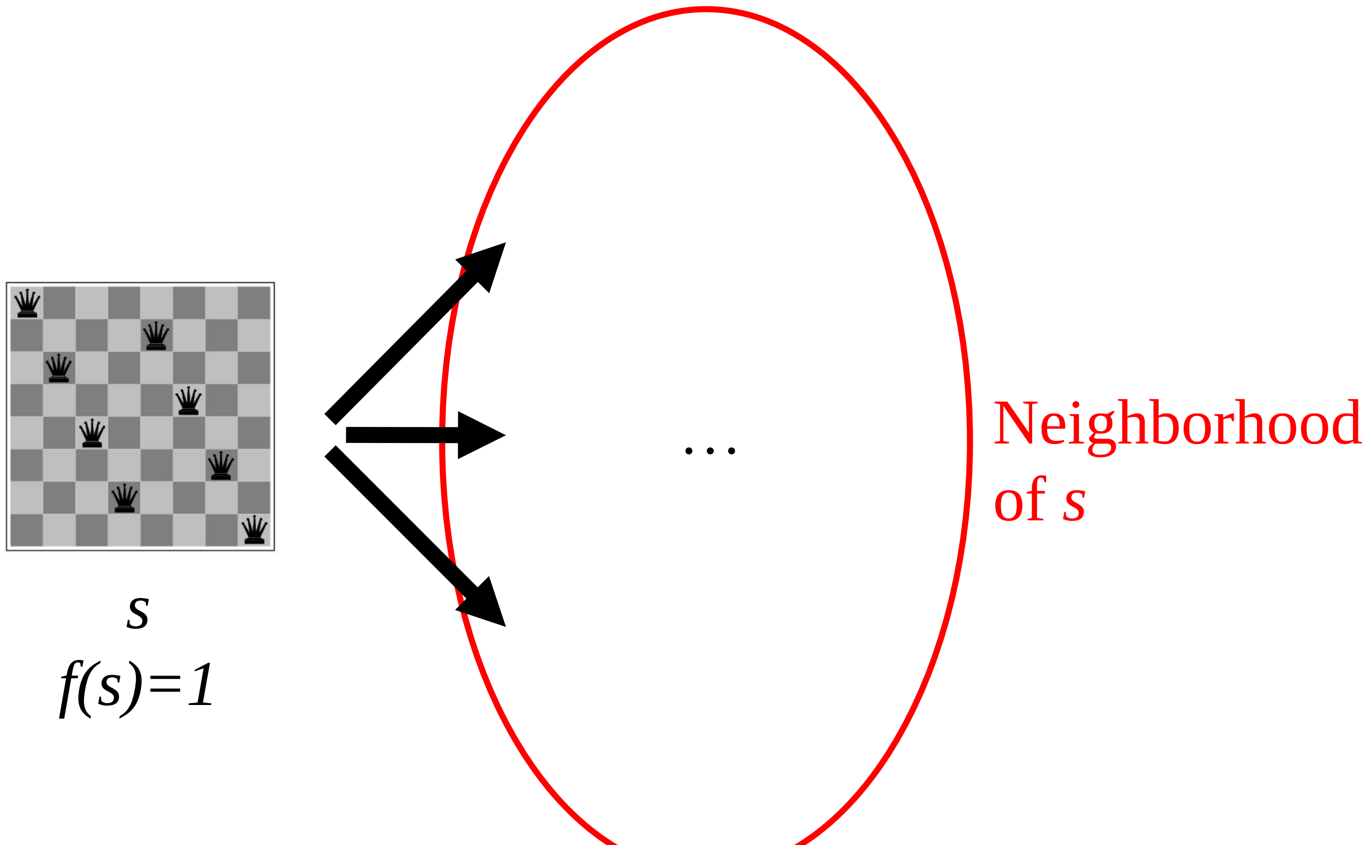
1. HILL CLIMBING

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

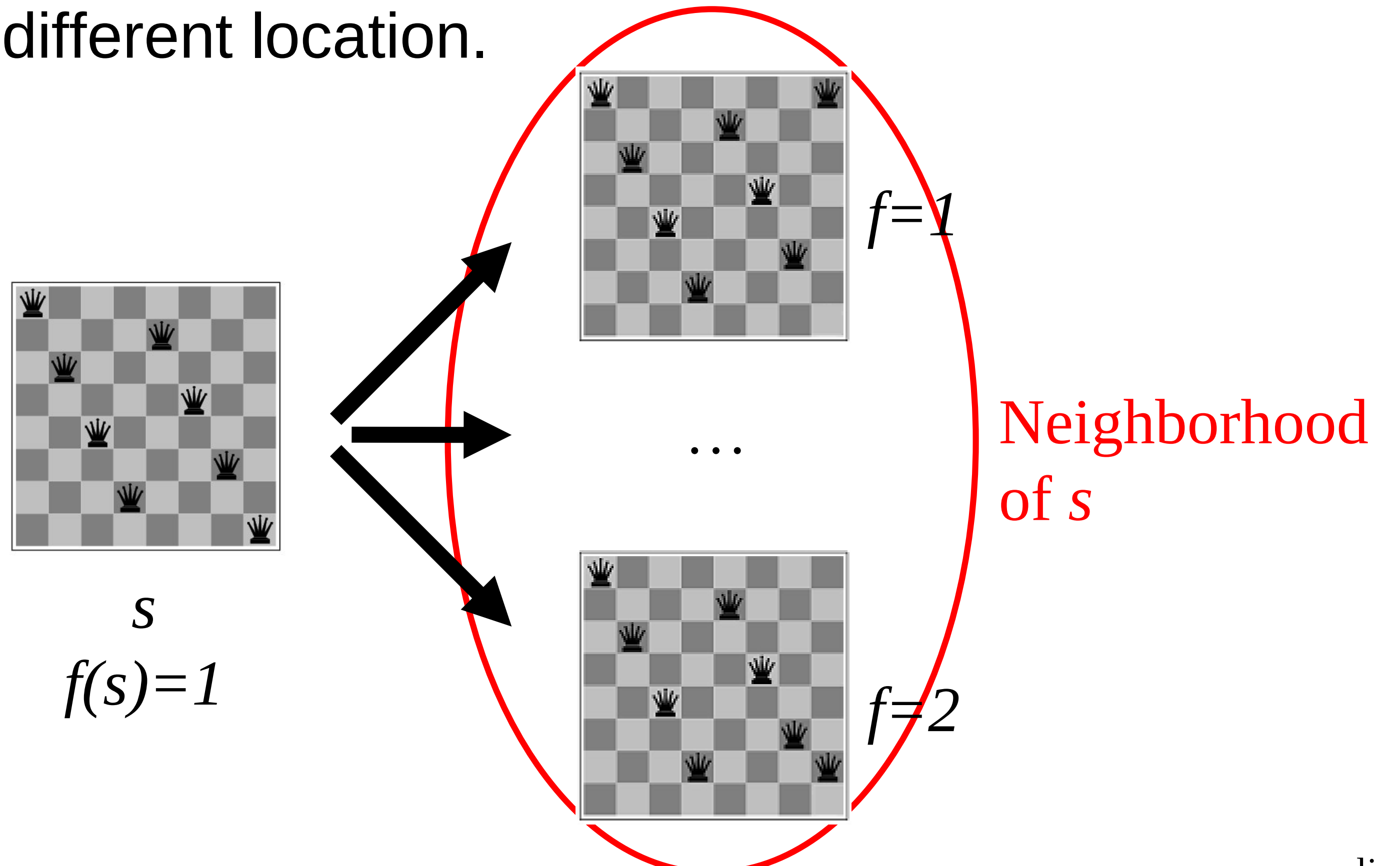
Neighbors: N-queen

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



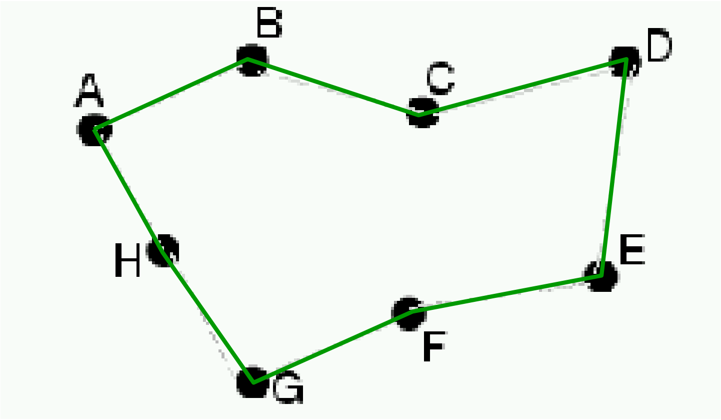
Neighbors: N-queen

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



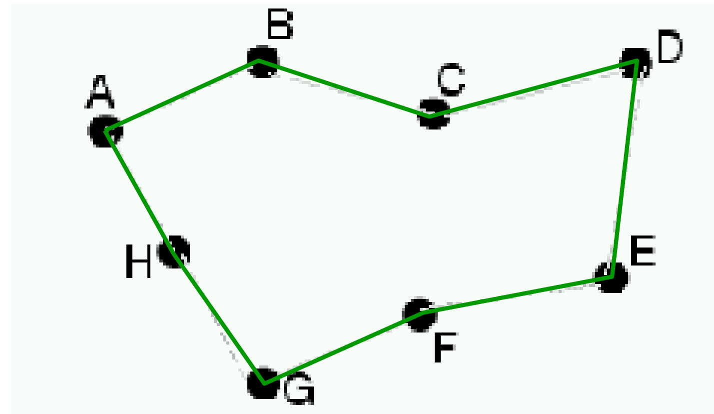
Neighbors: TSP

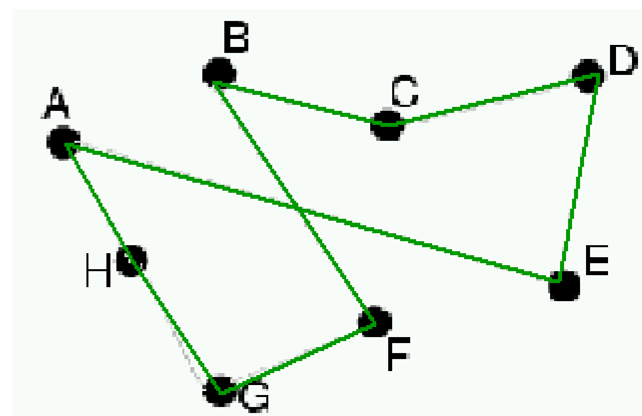
- state: A-B-C-D-E-F-G-H-A
- *f* = length of tour
- One possibility: 2-change

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

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.

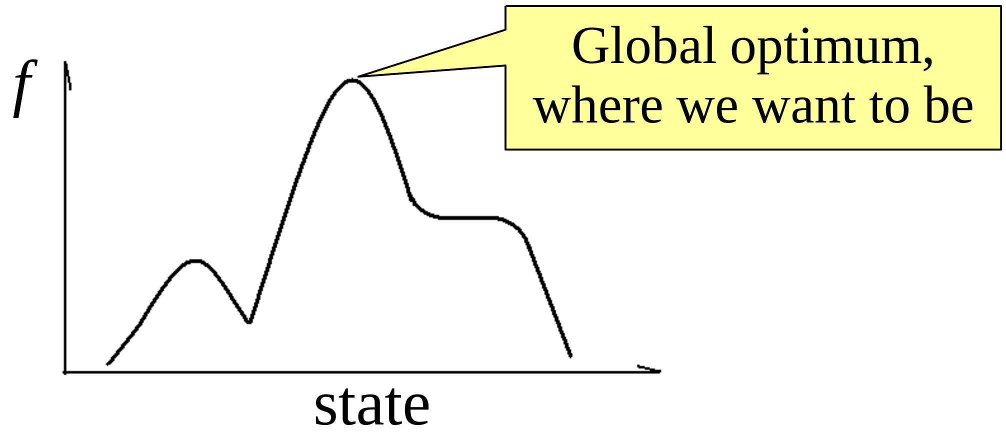
- Not the most sophisticated algorithm in the world.
- Very greedy.
- Easily stuck.

your enemy:

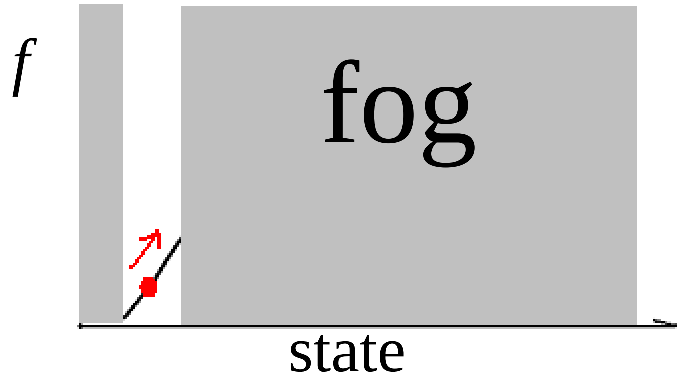
local
optima

Local optima in hill climbing

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

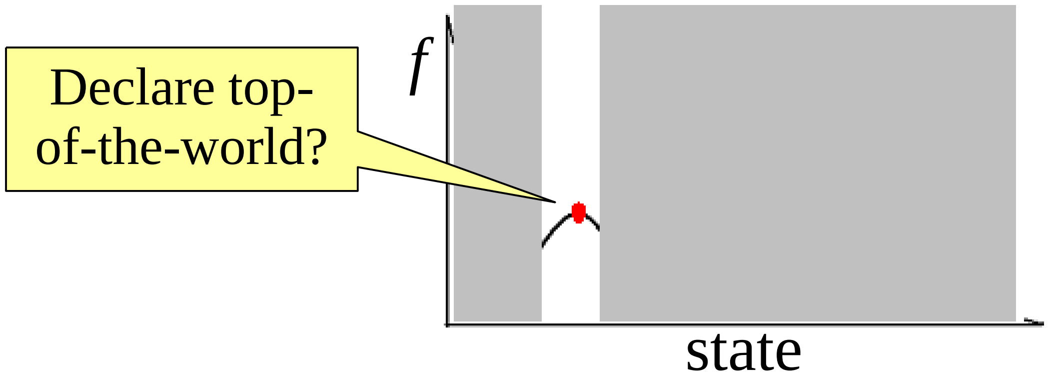


- But we can’t see the landscape all at once. Only see the neighborhood. Climb in fog.

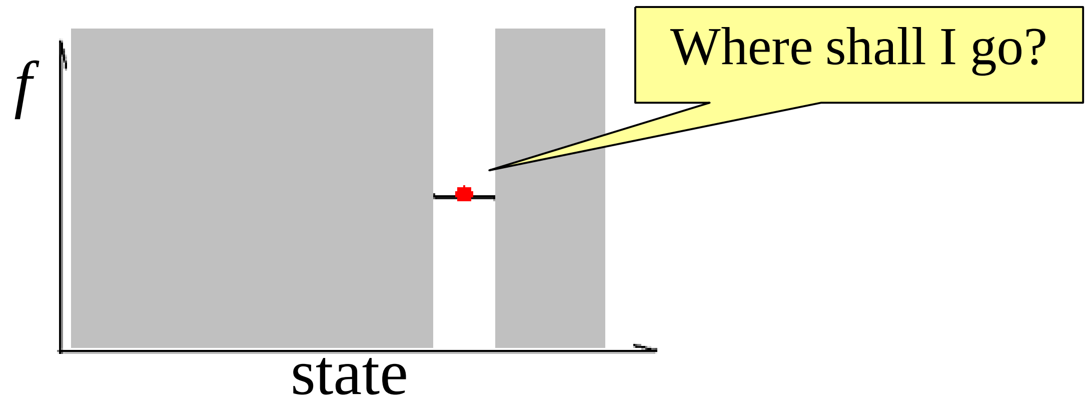


Local optima in hill climbing

- Local optima (there can be many!)



- Plateaux



Local optima in hill clmbing

- Local optima (there ca e ma

Declare t the wor

f

- Plate

ere shall I go?

e

The rest of the lecture is about

Escaping local optima

Not every local minimum should be escaped

Repeated 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 of hill climbing

- **Question**: How do we make hill climbing less greedy?

Variations 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 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 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?

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.

Variations of hill climbing

WALKSAT [Selman]

- Pick a random unsatisfied clause
- Consider 3 neighbors: flip each variable
- 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.

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