Local Search

CPSC 322 Lecture 13

Lecture Overview

- Recap solving CSP systematically
- Local search
- Constrained Optimization
- Greedy Descent / Hill Climbing: Problems

Systematically solving CSPs: Summary

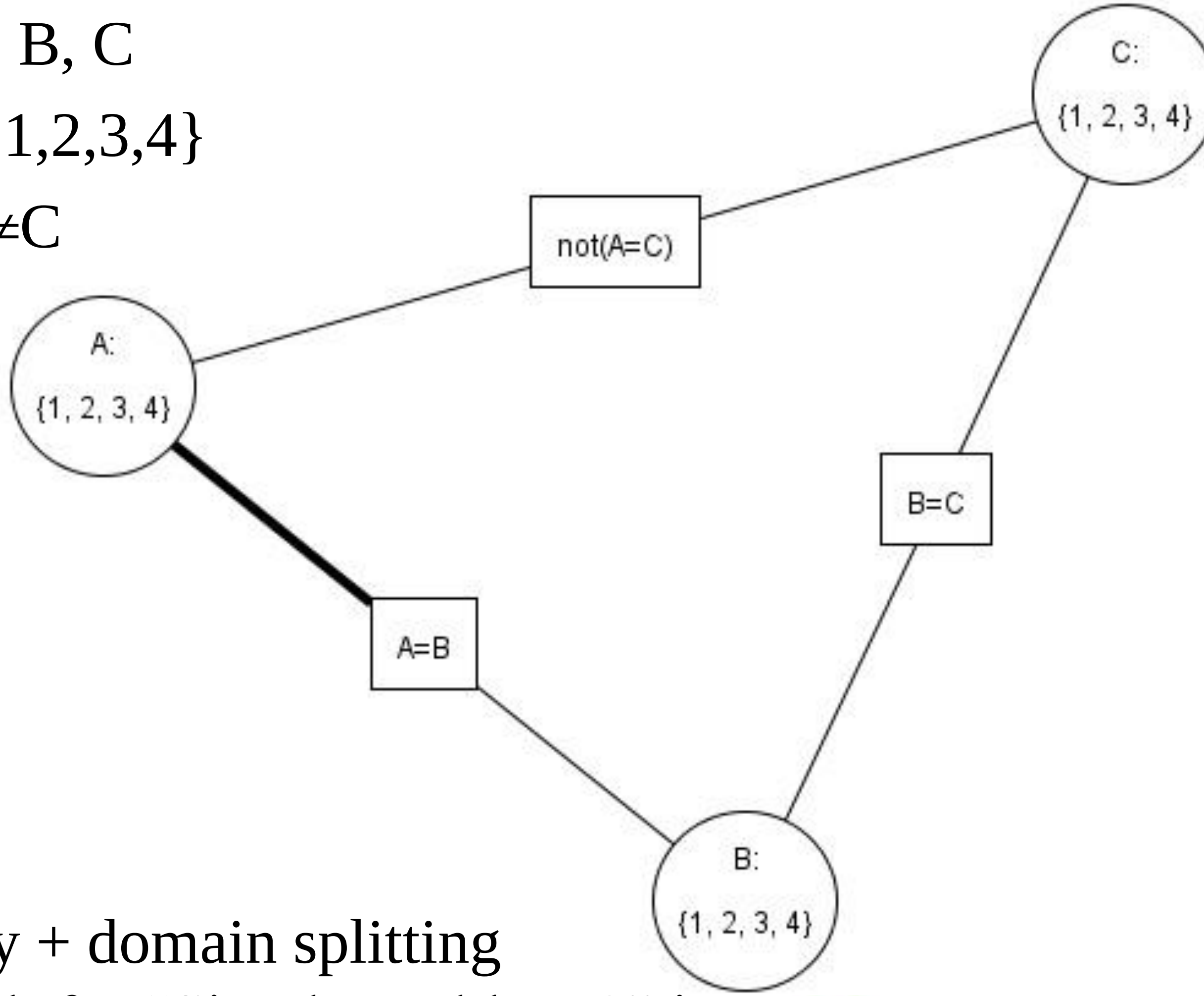
- Apply Depth-First Search with Pruning

OR

- Build Constraint Network
- Apply Arc Consistency
  - At least one domain is empty $\rightarrow$ **no solution**
  - Each domain has a single value $\rightarrow$ **one solution**
  - Some domains have more than one value $\rightarrow$ **???** SO
- Search by Domain Splitting
  - Split the problem into a number of disjoint cases
  - Apply Arc Consistency to each case
  - Repeat as needed

Domain Splitting in Action:

- 3 variables: A, B, C
- Domains: all {1,2,3,4}
- A=B, B=C, A≠C



- Let’s trace
  arc consistency + domain splitting
  for this network for “Simple Problem 2” in AIspace

Learning Goals for this class

You can:

- Implement local search for a CSP.
  - Implement different ways to generate neighbors
  - Implement scoring functions to solve a CSP by local search through either greedy descent or hill-climbing.

Lecture Overview

- Recap
- Local search
- Constrained Optimization
- Greedy Descent / Hill Climbing: Problems

Local Search motivation: Scale

- Many CSPs (scheduling, DNA computing, etc.) are simply too big for systematic approaches
- If you have 10^5 vars with $dom(var_i) = 10^4$

- Systematic Search

  $O(b^m)$ so $(10^4)^{(10^5)}$

- Arc Consistency

  A. $10^5 * 10^4$
  B. $10^{10} * 10^8$
  C. $10^{10} * 10^{12}$

- but if solutions are densely distributed…….

Local Search: General Method

Remember, for CSP a solution is a possible world, **not** a path

- Start from a possible world
- Generate some neighbors (“similar” possible worlds)
- Move from the current node to a neighbor, selected according to a particular strategy

Local Search: Selecting Neighbors

How do we determine the neighbors?

- Usually this is simple: some small incremental change to the variable assignment
  a) assignments that differ in one variable's value, by (for instance) a value difference of +1
  b) assignments that differ (by any amount) in one variable's value
  c) assignments that differ in two variables' values, etc.

Iterative Best Improvement

- How to determine the neighbor node to be selected?
- Iterative Best Improvement:
  - select the neighbor that optimizes some evaluation function
- Which strategy would make sense? Select neighbor with … *(choose the **best** answer, that is applicable in every situation)*

A. Maximal number of constraint violations

B. Similar number of constraint violations as current state

C. No constraint violations

D. Minimal number of constraint violations

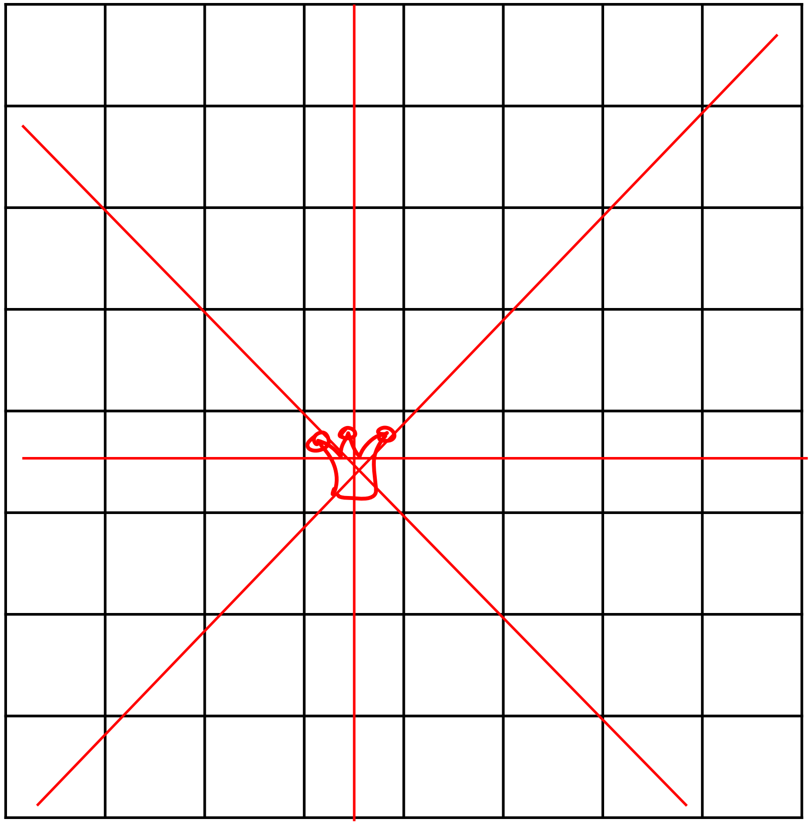
E. Vogon poetry (the 3rd worst in the universe)

Selecting the best neighbor

- Example: A,B,C same domain {1,2,3} , (A=B, A>1, C≠3)
- Suppose we start with {A=1, B=1, C=1}
  - What are the neighbors? *(depends on neighbor function)*
  - Which neighbor is the best?

Example: N-Queens

- Put n queens on an n × n board with no two queens on the same row, column, or diagonal (i.e attacking each other)



- Positions a queen can attack

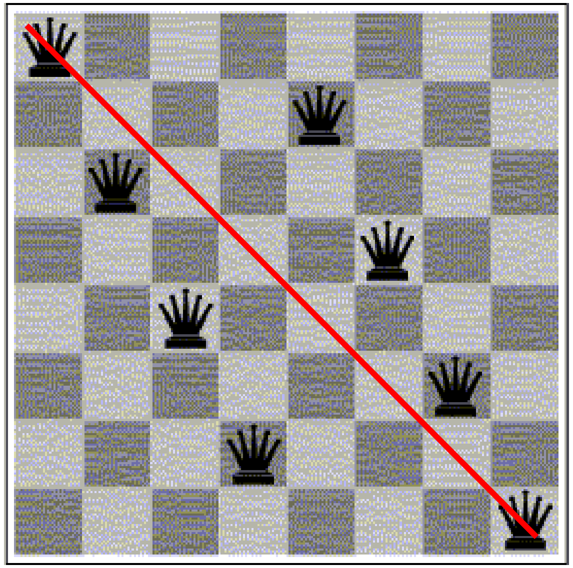
Example: N-queen as a local search problem

CSP: N-queen CSP

- One variable per column; domains {1,…,N} => row where the queen in the i^{th} column sits;
- Constraints: no two queens in the same row, column or diagonal

Neighbour relation: value of a single column differs

Scoring function: number of attacks



How many neighbors ?

A. 100
B. 80
C. 56
D. 8
E. 42

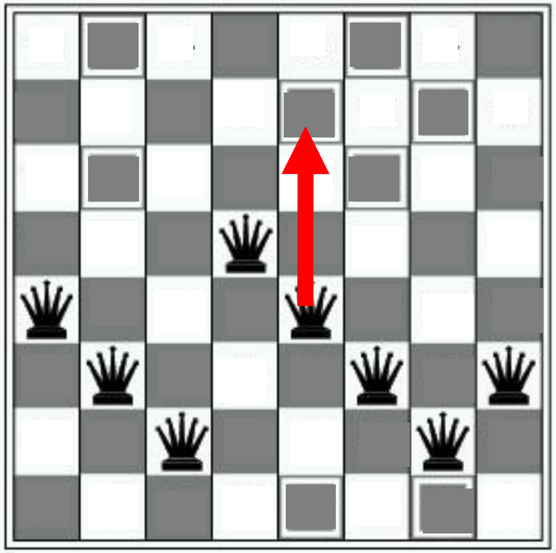
Example: Local Search for N-Queens

For each column, assign randomly each queen to a row
  (a number between 1 and N)

Repeat

- For each column & each number: Evaluate how many constraint violations changing the assignment would yield
- Choose the column and number that leads to the fewest violated constraints; change the assignment

Until solved



n-queens, Why?

Why this problem?

Lots of research in the 90's on local search for CSP was generated by the observation that the run-time of local search on n-queens problems is essentially **independent of problem size**!

Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10{,}000{,}000$)

Lecture Overview

- Recap
- Local search
- **Constrained Optimization**
- **Greedy Descent / Hill Climbing: Problems**

Constrained Optimization Problems

So far we have assumed that we just want to find a possible world that satisfies all the constraints.

But sometimes solutions may have different values / costs

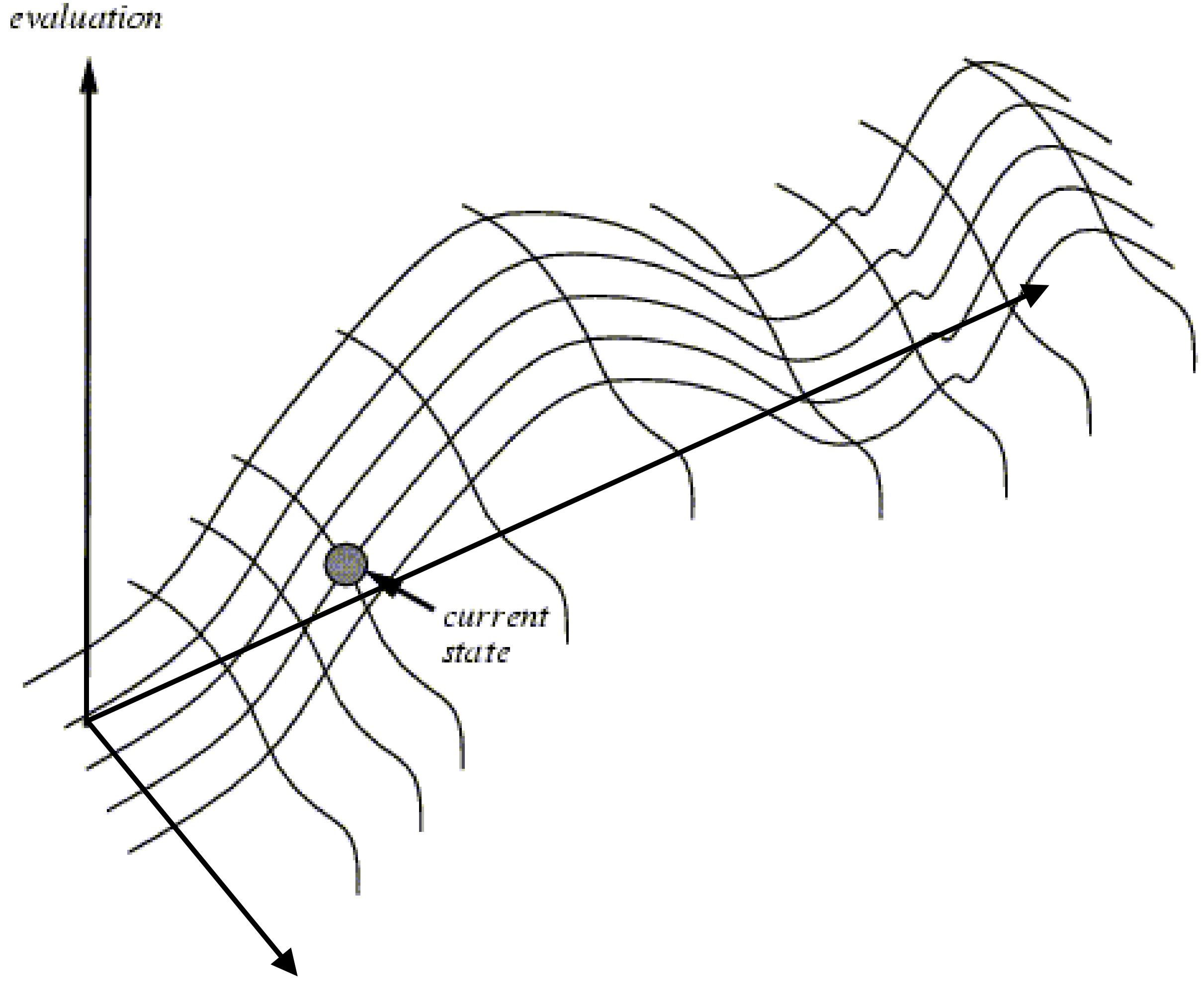
- We want to find the optimal solution that
  - **maximizes the value** or
  - **minimizes the cost**

Hill Climbing means selecting the neighbor which best improves a (value-based) scoring function.

Greedy Descent means selecting the neighbor which minimizes a (cost-based) scoring function.

Hill Climbing

NOTE: Everything that will be said for Hill Climbing is also true for Greedy Descent



Constrained Optimization Example

Example: A,B,C same domain {1,2,3} , (A=B, A>1, C≠3)

- Value = (C+A) so we want a solution that maximizes that

The scoring function we'd like to maximize might be:

f(n) = (C + A) + #-of-satisfied-const

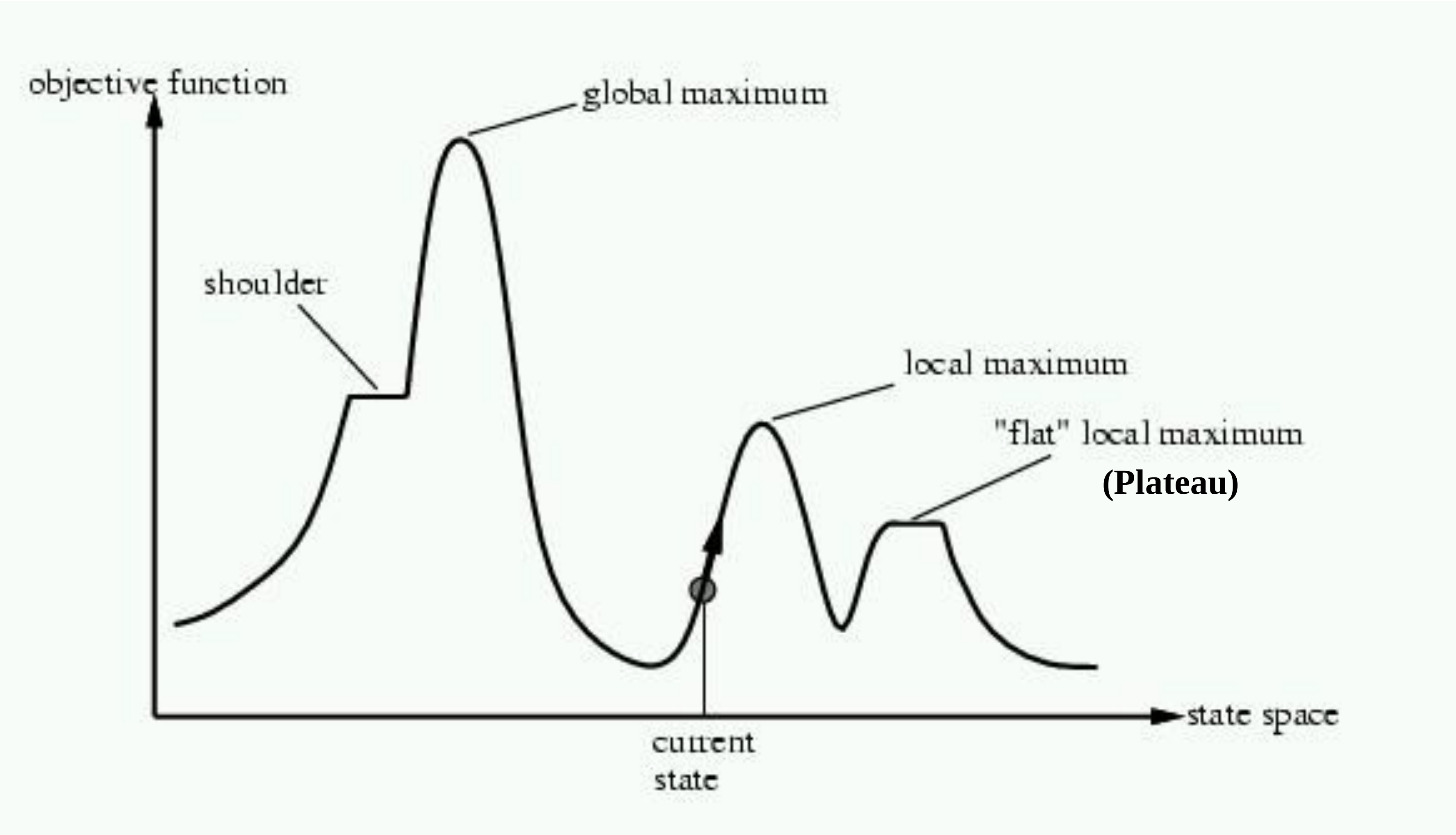
If we're doing Greedy Descent, then what we want to minimize is cost + *#-of-conflicts*

Lecture Overview

- Recap
- Local search
- **Constrained Optimization**
- **Greedy Descent / Hill Climbing: Problems**

Problems with Hill Climbing

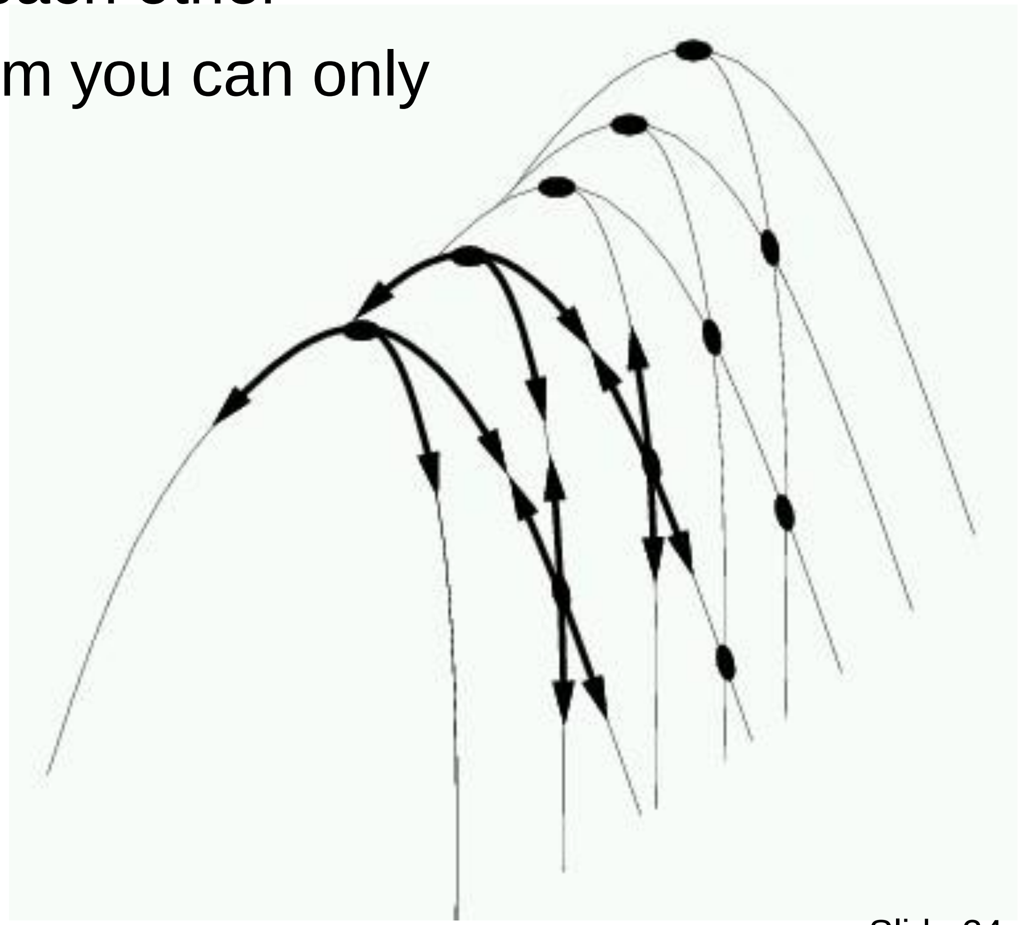
- Local Maxima
- Plateaus and Shoulders



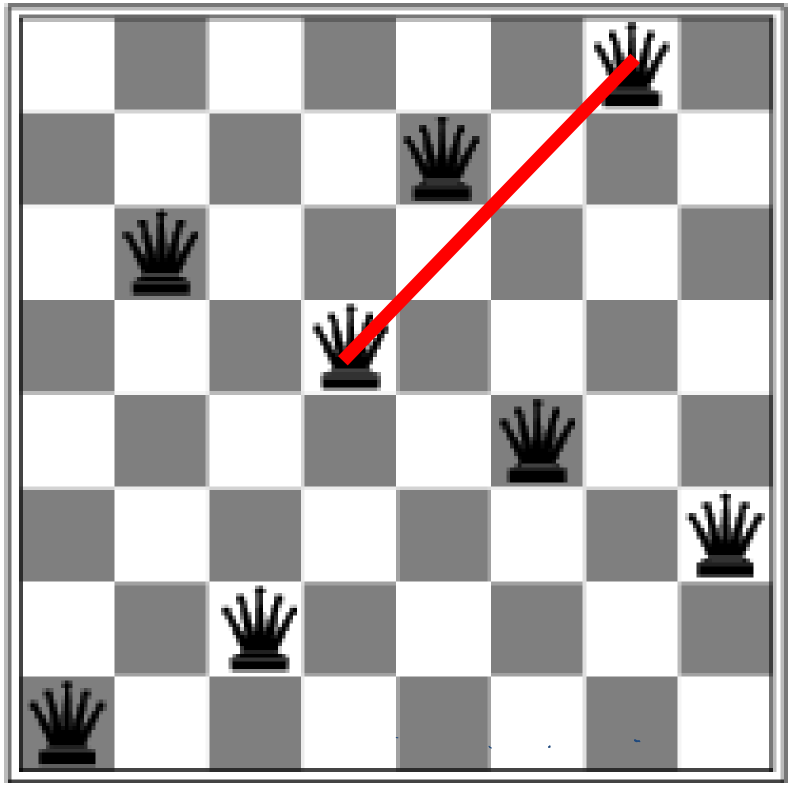
More problems in higher dimensions

E.g., Ridges – sequence of local maxima not directly connected to each other

From each local maximum you can only go downhill



Corresponding problem for GreedyDescent Local minimum example: 8-queens problem



A local minimum with *h = 1*

(all neighbors have h > 1)

Local Search: Summary

- A useful method for large CSPs
  - Start from a possible world (often randomly chosen)
  - Generate some neighbors (“similar” possible worlds)
  - Move from current node to a neighbor, selected to minimize/maximize a scoring function which combines:
    - Information about how many constraints are violated
    - Information about the cost/quality of the solution (you want the best solution, not just a solution)

Next Class

- How to address problems with Greedy Descent / Hill Climbing?

Stochastic Local Search (SLS)