Local Search

Computer Science cpsc322, Lecture 14

(Textbook Chpt 4.8)

May, 30, 2017

Announcements

- Assignment1 due now!

- Assignment2 out today

Lecture Overview

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

Systematically solving CSPs: Summary

- Build Constraint Network
- Apply Arc Consistency
  - One domain is empty → no sol
  - Each domain has a single value → unique sol
  - Some domains have more than one value → ?!

    may or may not have a solution

- Apply Depth-First Search with Pruning
- Search by Domain Splitting
  - Split the problem in a number of disjoint cases
  - Apply Arc Consistency to each case

Lecture Overview

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

Local Search motivation: Scale

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

- Systematic Search

  b^m

  $(10^4)^{10^5}$

- Arc Consistency

  iclicker.

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

  $n^2 d^3$

- 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
  - Example: A,B,C same domain {1,2,3}

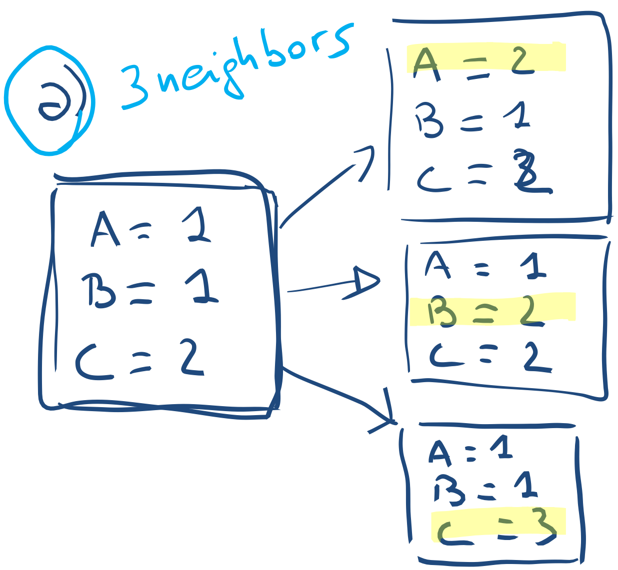
neighbors of start

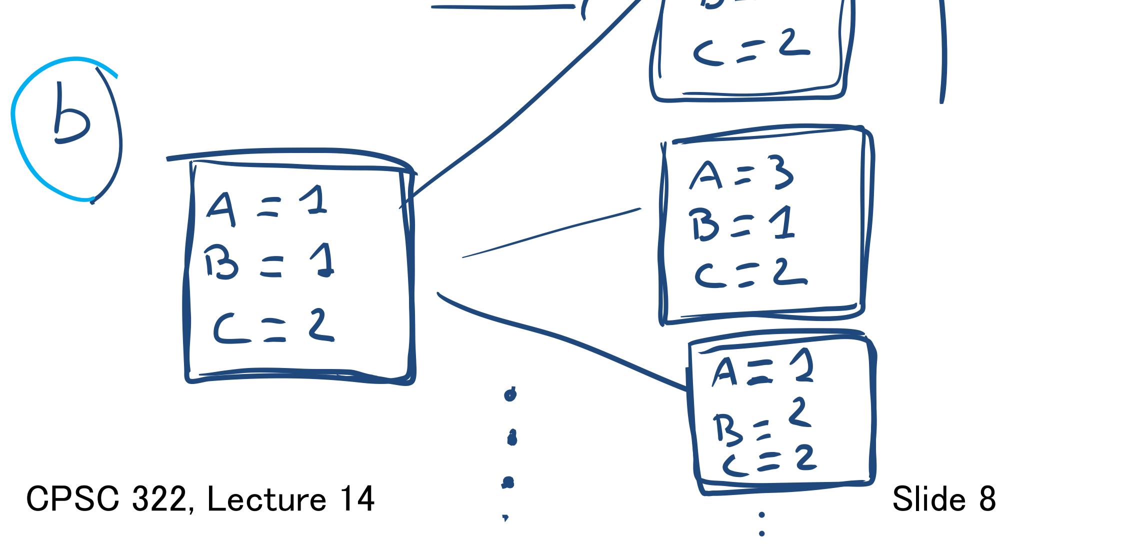
Local Search: Selecting Neighbors

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 in one variable's value
  - c) assignments that differ in two variables' values, etc.

- Example: A,B,C same domain {1,2,3}





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 …

A. Maximal number of constraint violations

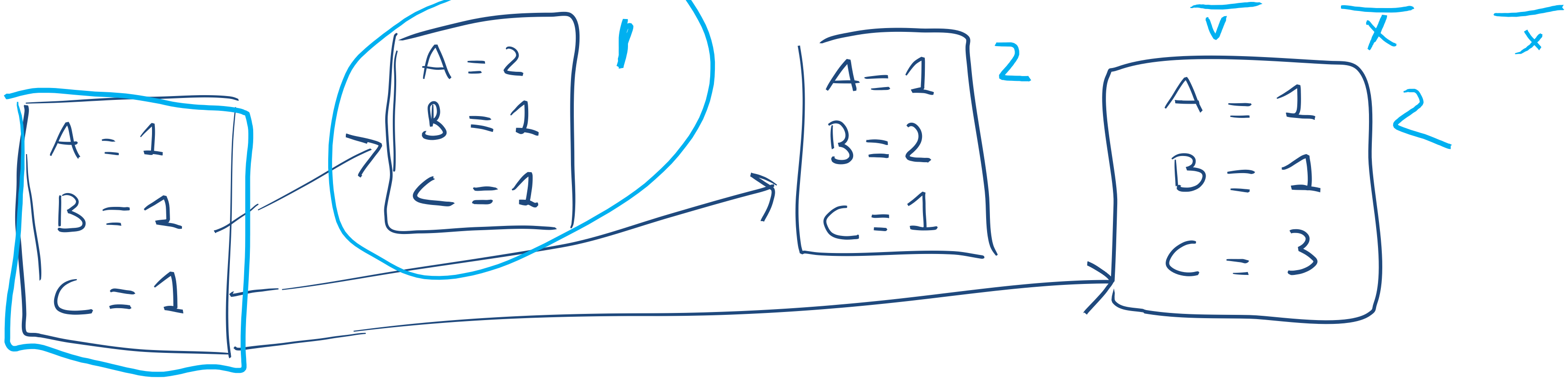
B. Similar number of constraint violations as current state

C. No constraint violations

D. Minimal number of constraint violations

Selecting the best neighbor

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

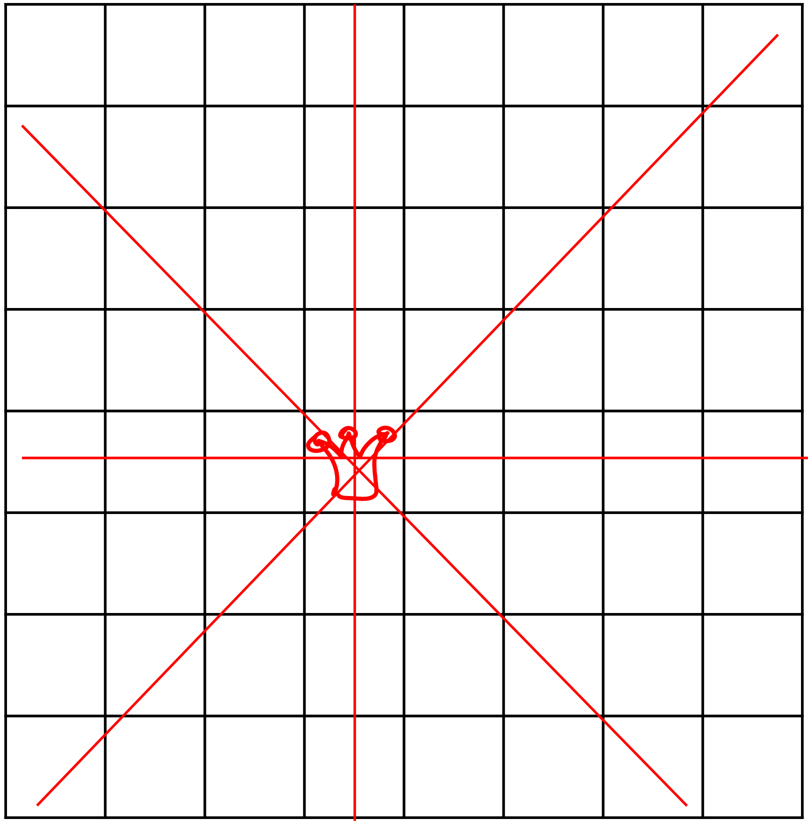


A common component of the scoring function (heuristic) =>
select the neighbor that results in the ……

– the min conflicts heuristics

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

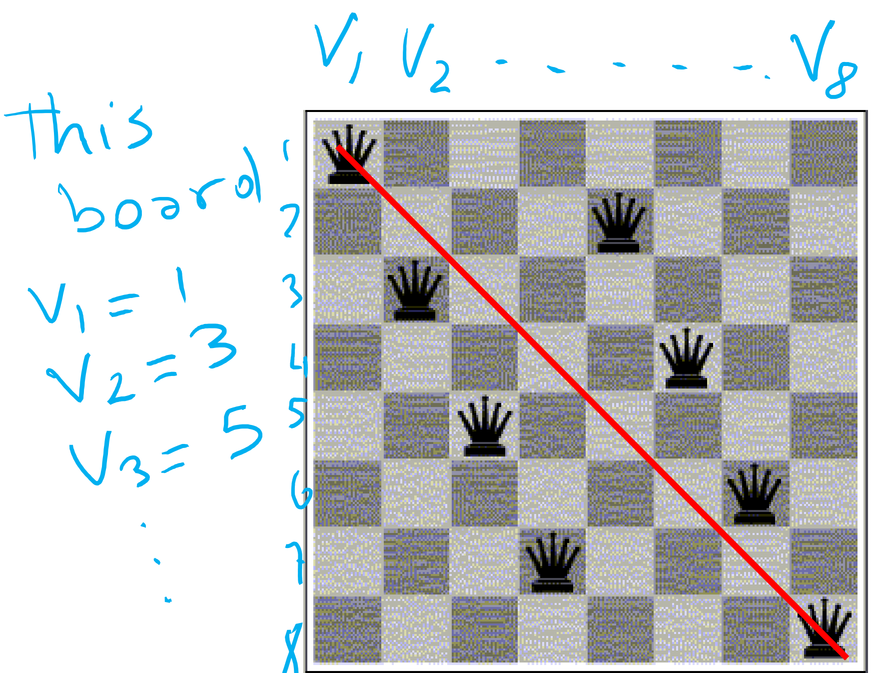
Example: N-queen as a local search problem

CSP: N-queen CSP

- One variable per column; domains {1,⋯,N} => row where the queen in the i^{th} column seats;
- 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. 90

C. 56

D. 9

$N * (N-1)$

$8 * 7$

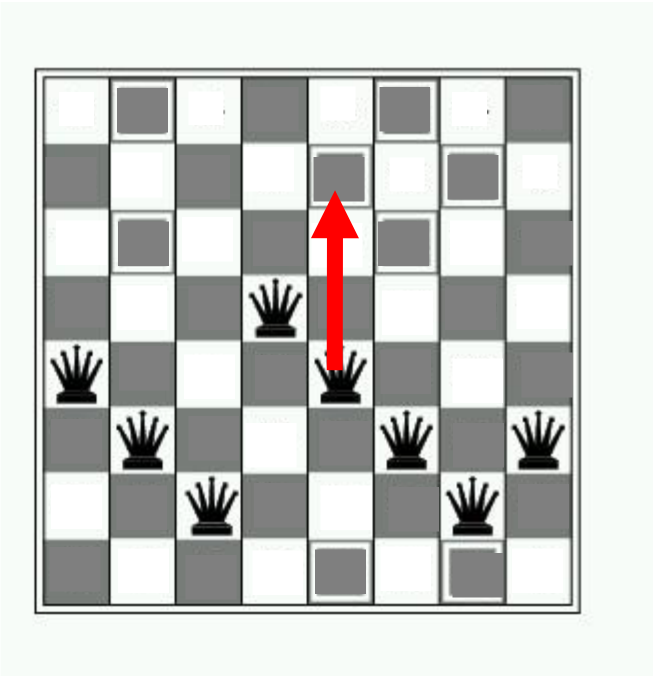
Example: Greedy descent for N-Queen

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

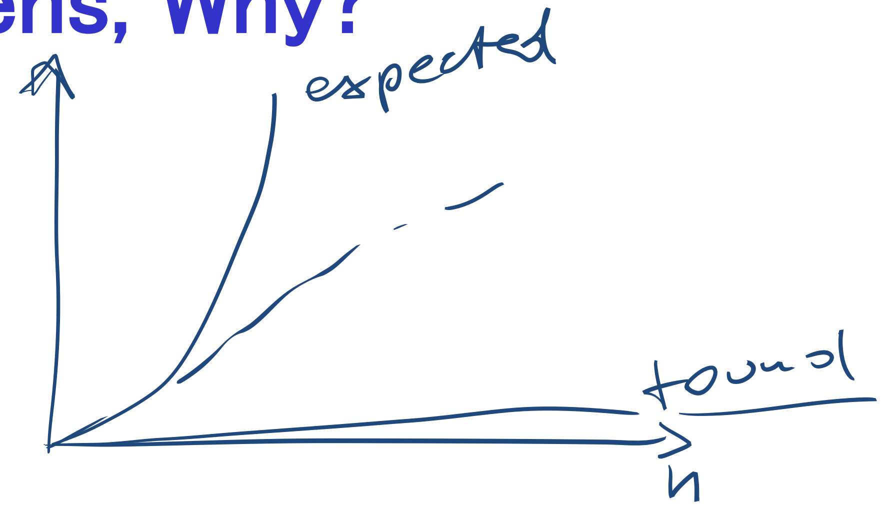
Repeat

- 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 it

Until solved



n–queens, Why?



Why this problem?

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

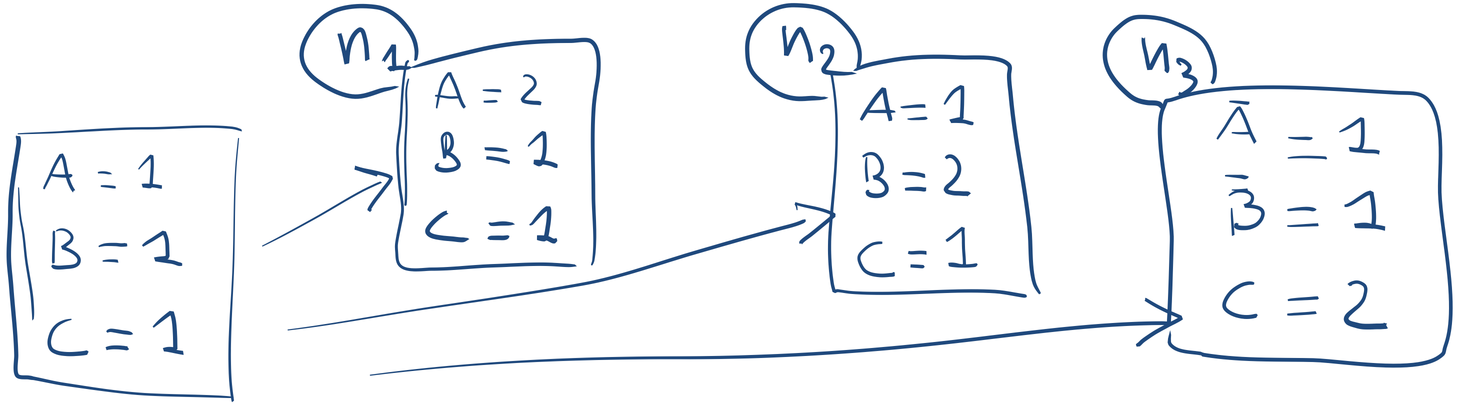
So far we have assumed that we just want to find a possible world that satisfies all the constraints.

But sometimes solutions may have different values / costs

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

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 maximize that



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

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

n1: (1+2)+2 n2: (1+1)+1 n3: (2+1)+2

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. cost + # of conflicts

Lecture Overview

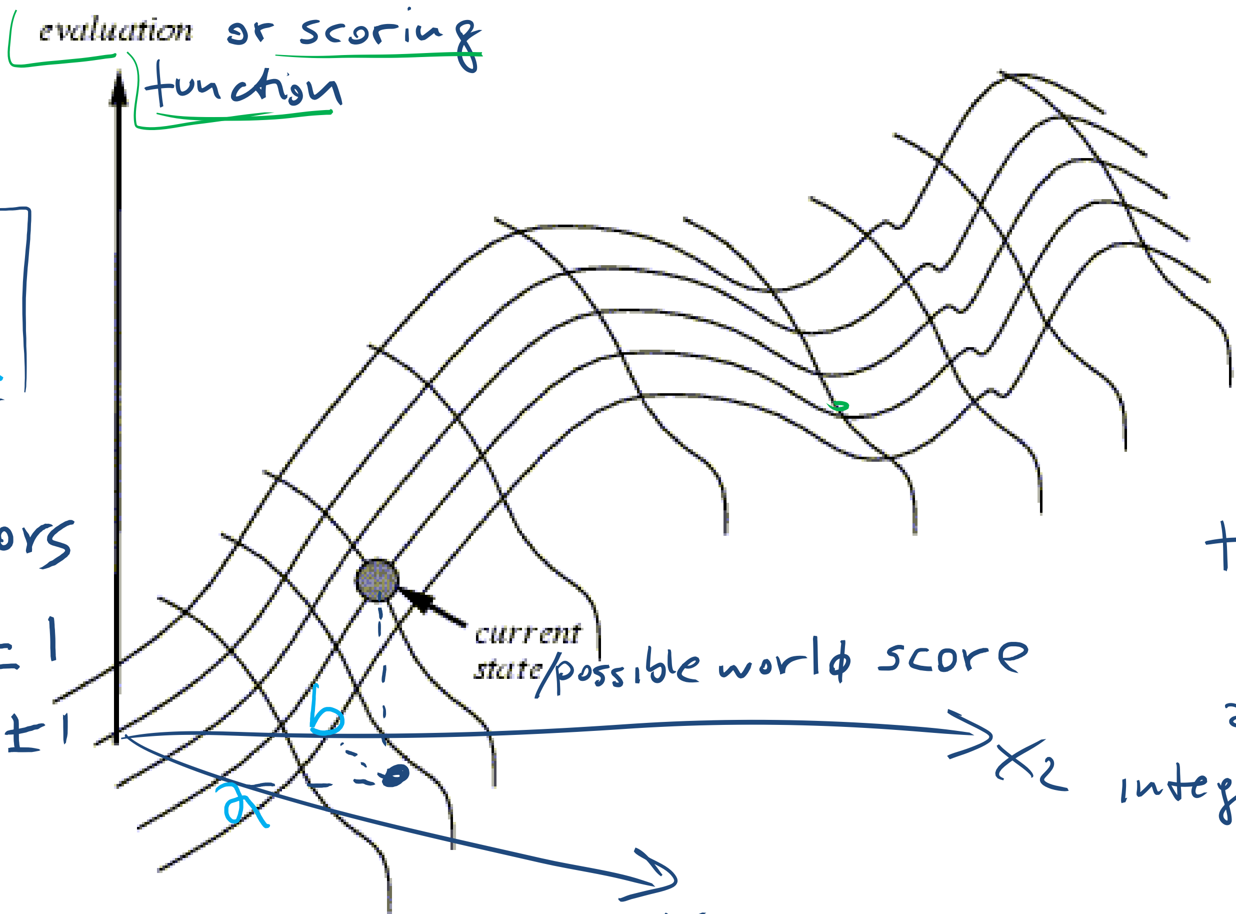
- Recap
- Local search
- Constrained Optimization
- **Greedy Descent / Hill Climbing: Problems**

Hill Climbing

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

current
$X_1 = a$
$X_2 = b$

4 neighbors
$X_1 = a \pm 1$
$X_2 = b \pm 1$

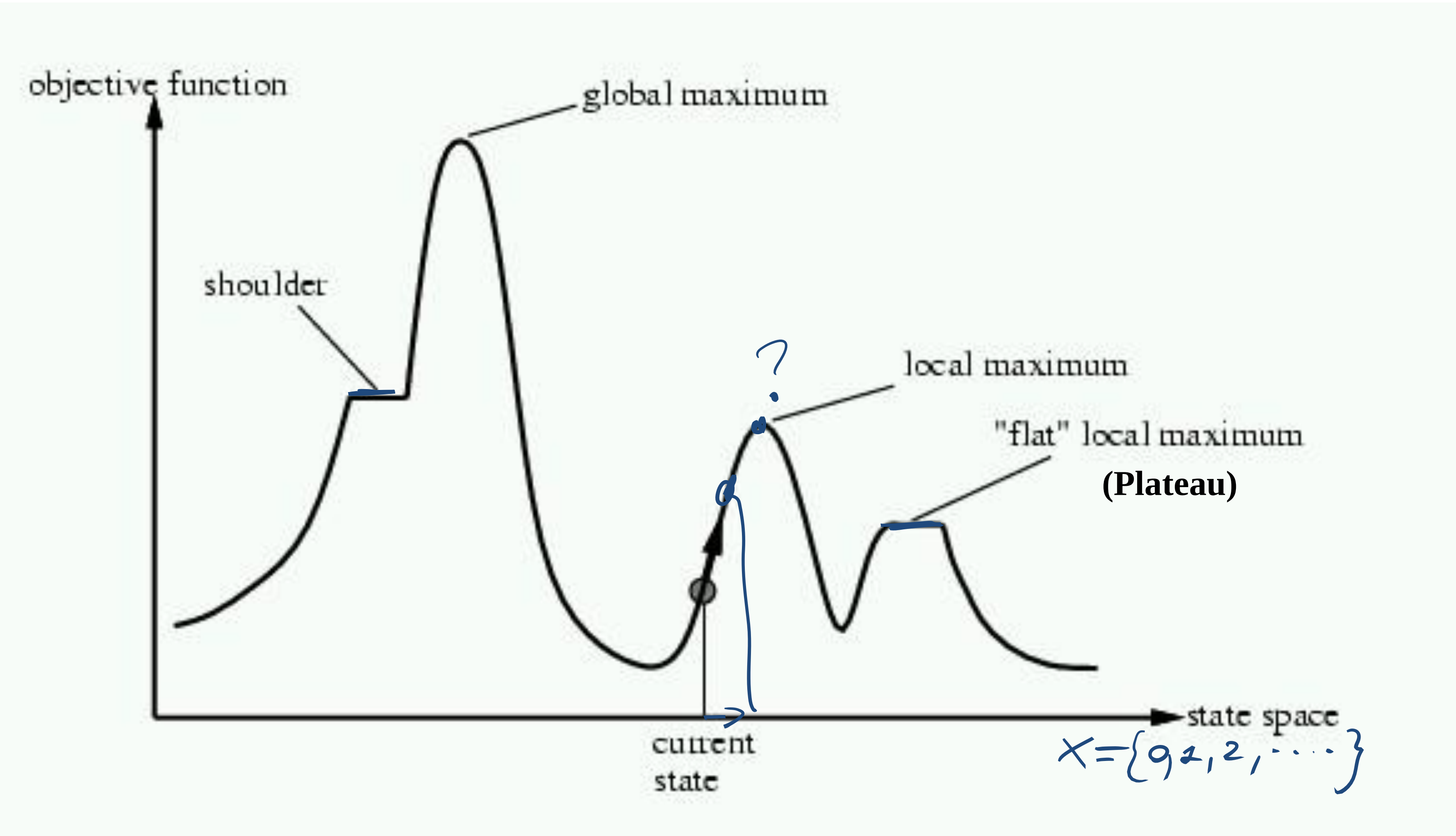


two vars
X_1 X_2
assume domain
integer [0, 1, 2 . . .]

Problems with Hill Climbing

Local Maxima.

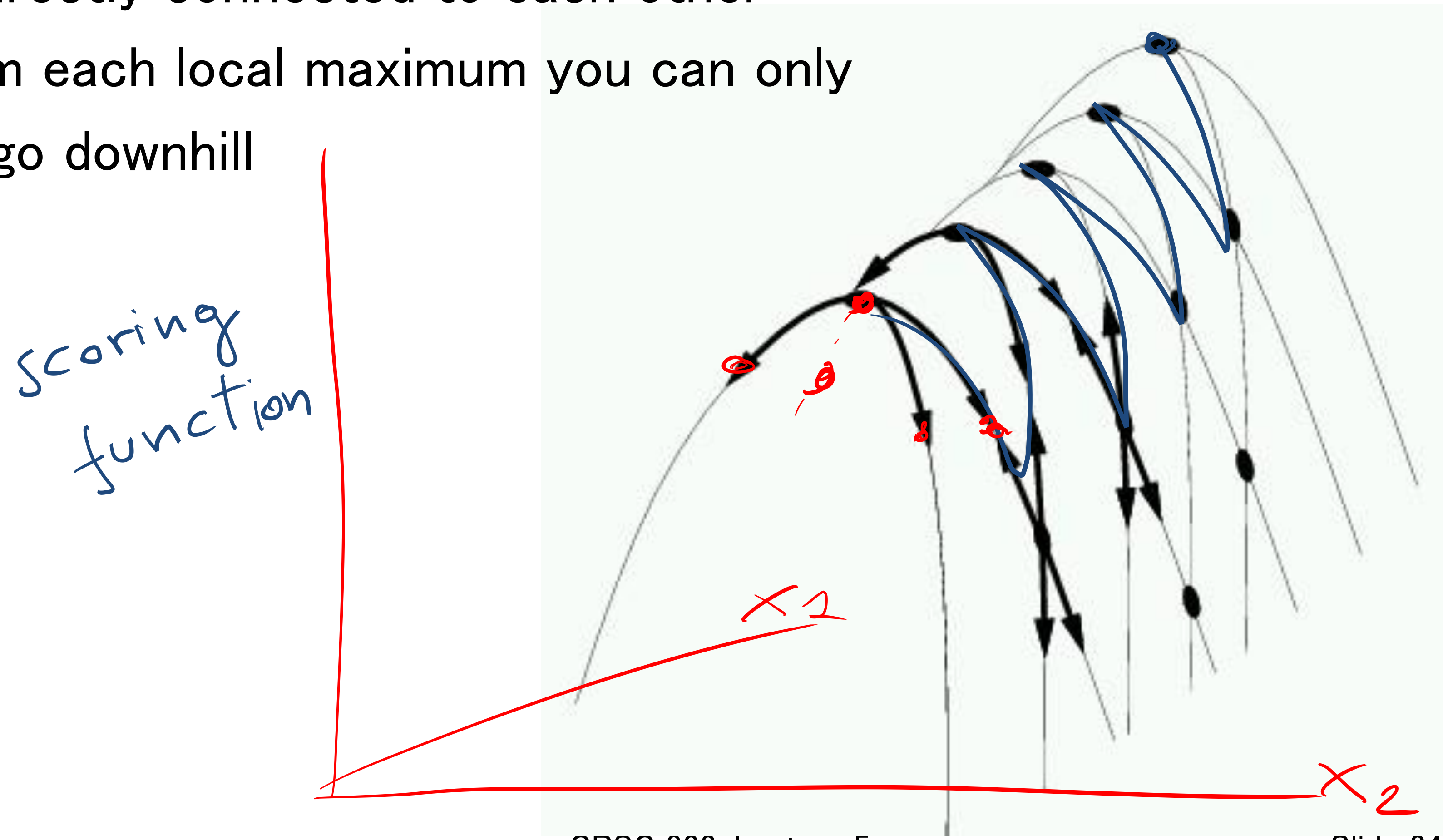
Plateau – Shoulders



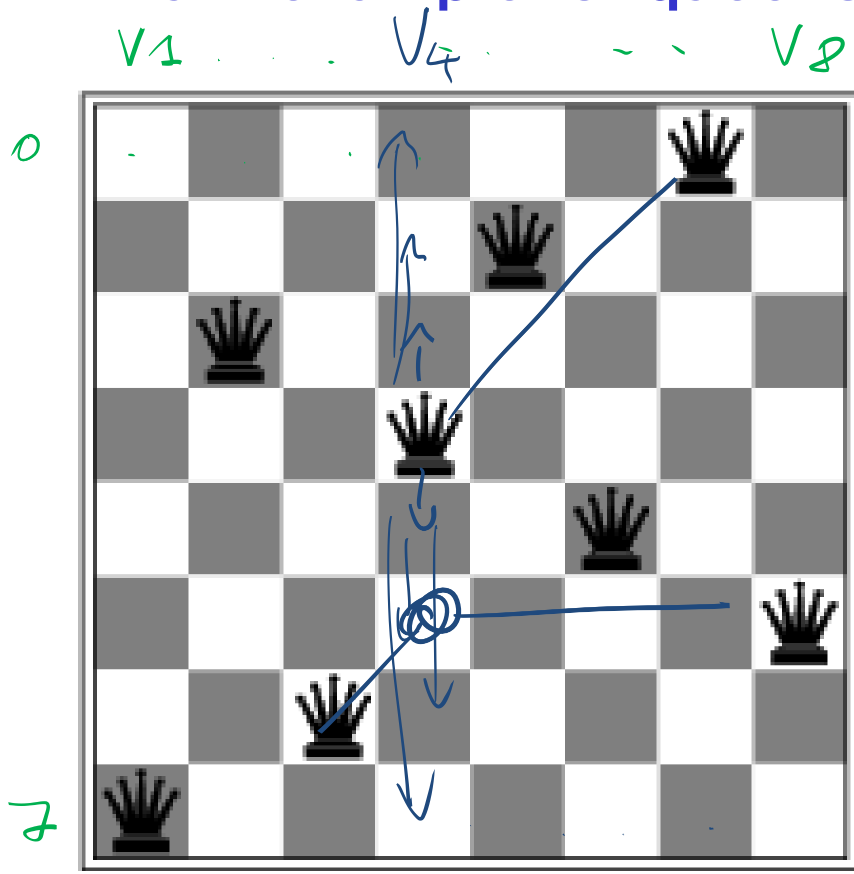
Even more Problems in higher dimensions

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



for all the moves (neighbors) h > 1

h = 0 for solution

A local minimum with $h = 1$

Local Search: Summary

- A useful method for large CSPs
  - Start from a possible world (randomly chosen)
  - Generate some neighbors ("similar" possible worlds)

    e.g. differ from current poss. world only by one variable's value
  - Move from current node to a neighbor, selected to minimize/maximize a scoring function which combines:
    - ✓ Info about how many constraints are violated
    - ✓ Information about the cost/quality of the solution (you want the best solution, not just a solution)

Learning Goals for today's 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.

Next Class

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

Stochastic Local Search (SLS)