Local Search

Computer Science cpsc322, Lecture 14

(Textbook Chpt 4.8)

Oct, 7, 2013

Department of Computer Science
Undergraduate Events
More details @ https://www.cs.ubc.ca/students/undergrad/life/upcoming-events

Global Relay Info Session/Tech Talk

Date: Mon., Oct 7
Time: 5:30 pm
Location: DMP 301

Amazon Info Session/Tech Talk

Date: Tues., Oct 8
Time: 5:30 pm
Location: DMP 110

Go Global Experience Fair

Date: Wed., Oct 9
Time: 11 am – 5 pm
Location: Irving K. Barber Learning Centre

Samsung Info Session

Date: Wed., Oct 9
Time: 11:30 am – 1:30 pm
Location: McLeod Rm 254

Google Info Session/Tech Talk

Date: Thurs., Oct 10
Time: 5:30 pm
Location: DMP 110

Announcements

- Assignment1 due now!
- Assignment2 out next week

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 maynot 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 $\mathrm{dom}(var_i) = 10^4$

- 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…….

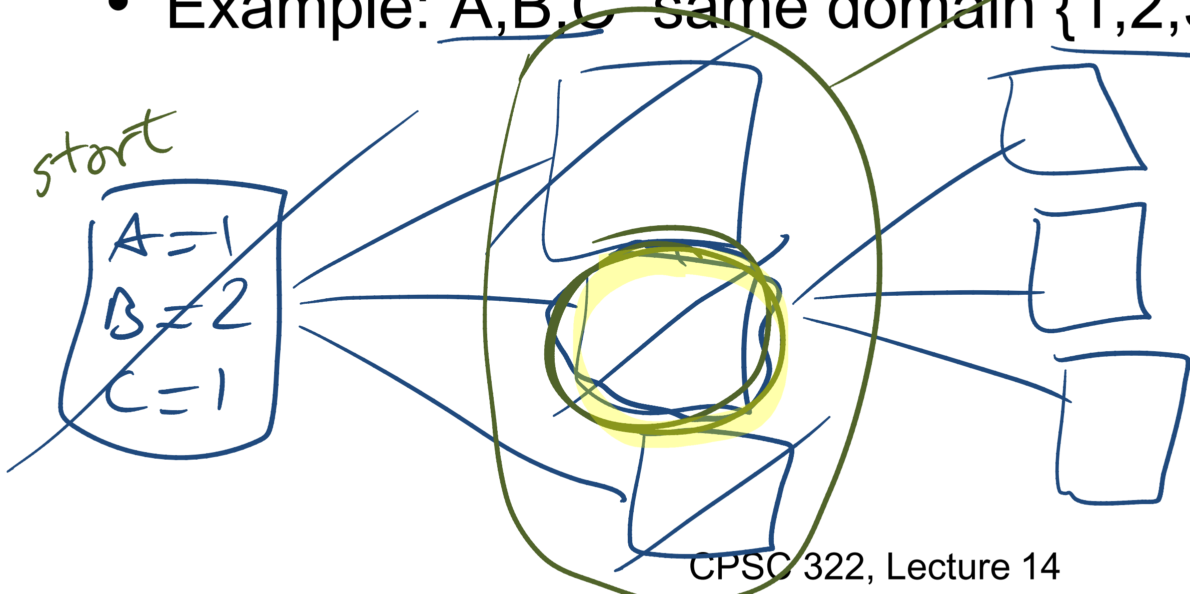
Local Search: General Method

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

neighbors of start

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



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.

c → 12 neighbors

6 neighbors

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

a) 3 neighbors

A = 1, B = 1, C = 2 → A = 2, B = 1, C = 2

A = 1, B = 1, C = 2 → A = 1, B = 2, C = 2

A = 1, B = 1, C = 2 → A = 1, B = 1, C = 3

b)

A = 1, B = 1, C = 2 → A = 2, B = 1, C = 2

A = 1, B = 1, C = 2 → A = 3, B = 1, C = 2

A = 1, B = 1, C = 2 → A = 1, B = 2, C = 2

⋮

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

Iterative Best Improvement

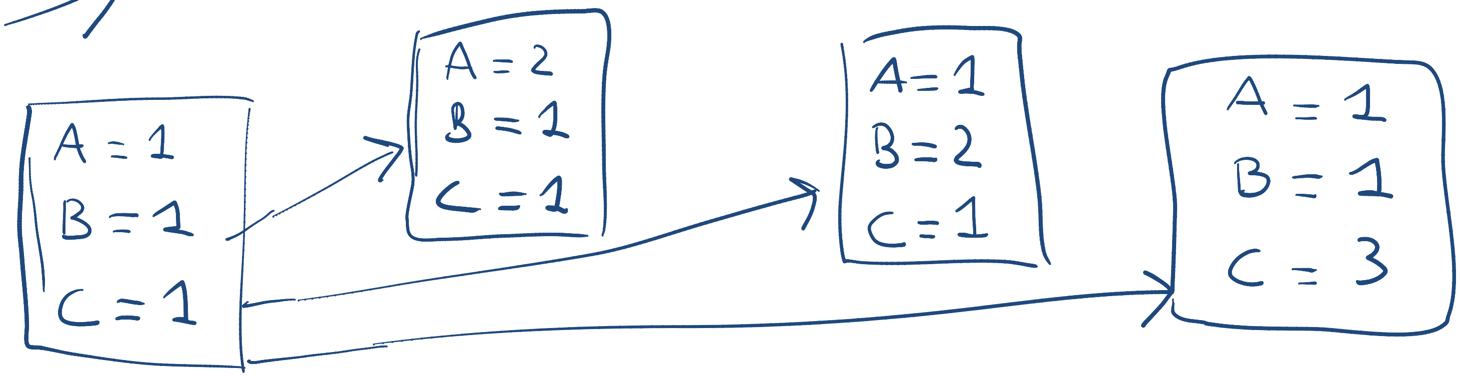
- How to determine the neighbor node to be selected?
- Iterative Best Improvement:
  - select the neighbor that optimizes some evaluation function
- Which strategy would make sense? Select

Minimal number of constraint violations

- Evaluation function:
  h(n): number of constraint violations in state n
- Greedy descent: evaluate h(n) for each neighbour, pick the neighbour n with minimal h(n)
- Hill climbing: equivalent algorithm for maximization problems
  - Here: maximize the number of constraints satisfied

Selecting the best neighbor

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



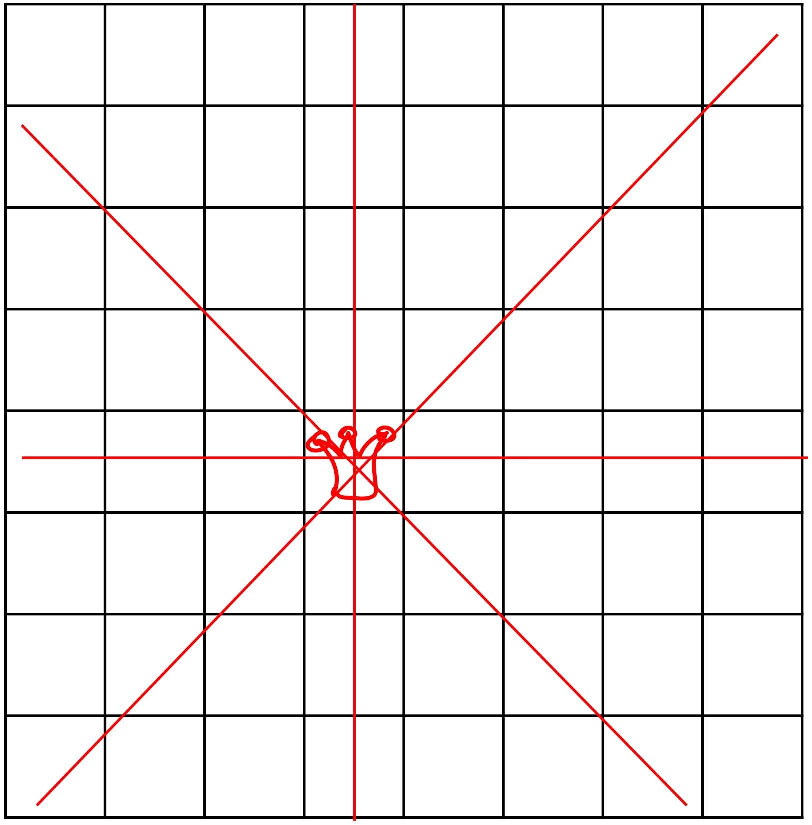
A common component of the scoring function (heuristic) =>
select the neighbor that results in the ……

- 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

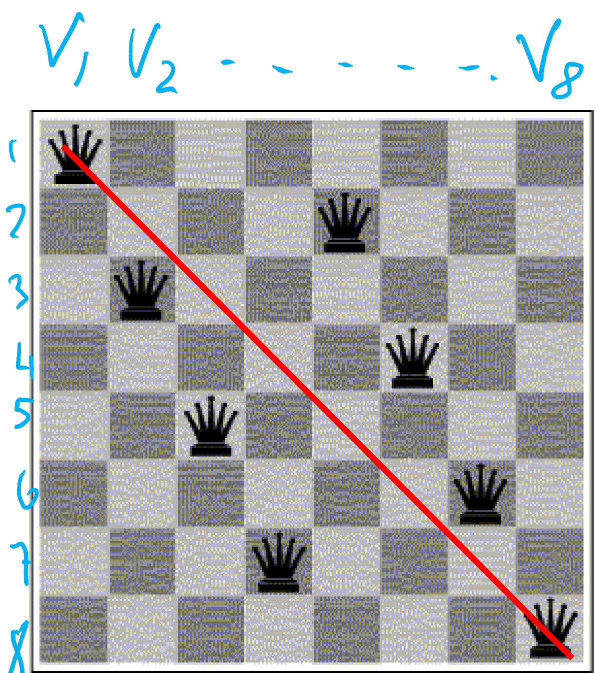
CSP: N-queen CSP

- 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

This board:
$V_1 = 1$
$V_2 = 3$
$V_3 = 5$
⋮



How many neighbors ?

A. 100
B. 90
C. 200
D. 9

ops! right answer is $7 * 8 = 56$

Always $(N-1) * N$

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)

Example: 4-Queens

~~States~~: 4 queens in 4 columns ($4^4 = 256$ states)

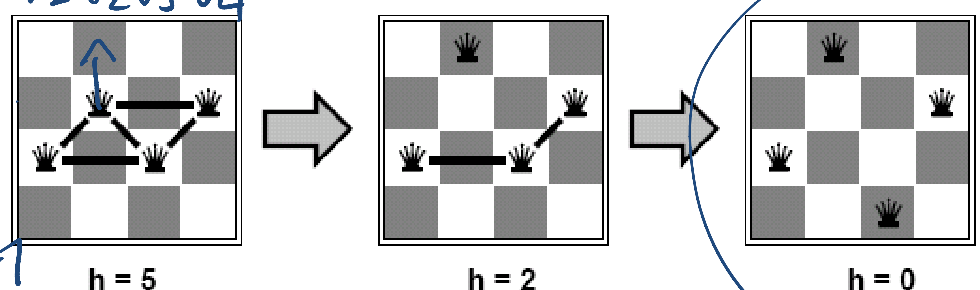
Operators: move queen in column (to generate neighbors)

Goal test: no attacks

Evaluation: $h(n)$ = number of attacks

V1 V2 V3 V4

0
1
2
3



solution

$V_1 = 2$
$V_2 = 1$
$V_3 = 2$
$V_4 = 1$

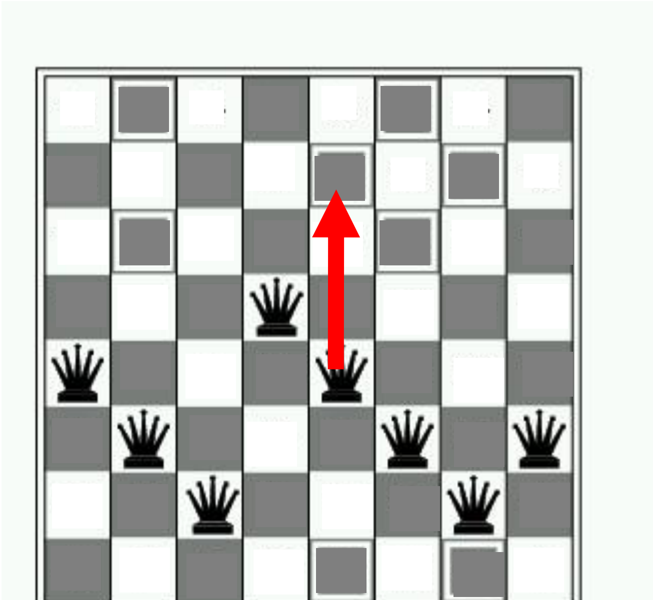
Example: Greedy descent for N-Queen

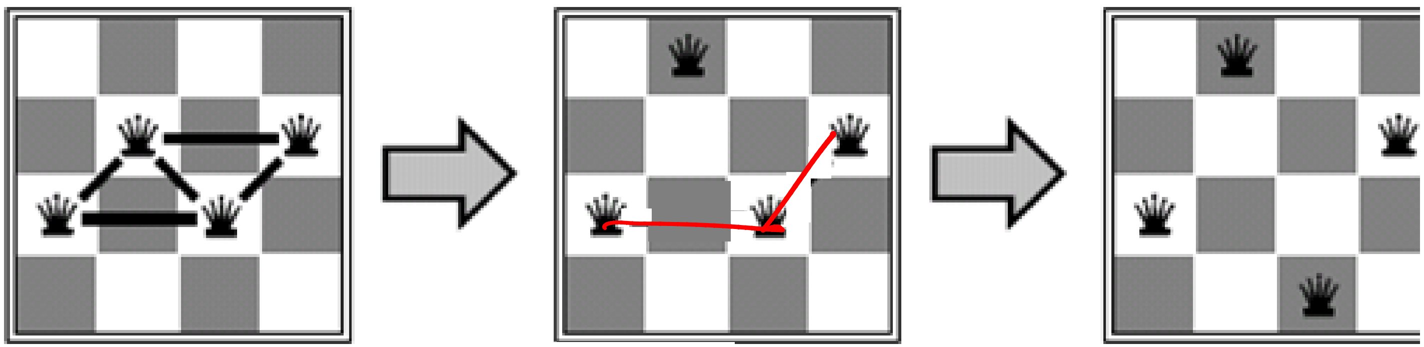
For each column, assign randomly each queen to a row
  (a number between 1 and N)

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

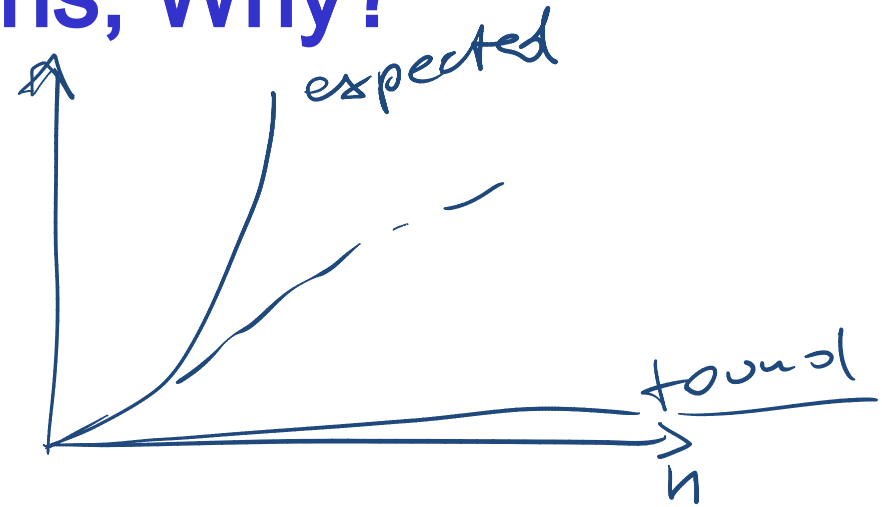




h = 5 h = ? h = ?

1 0 2 3

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

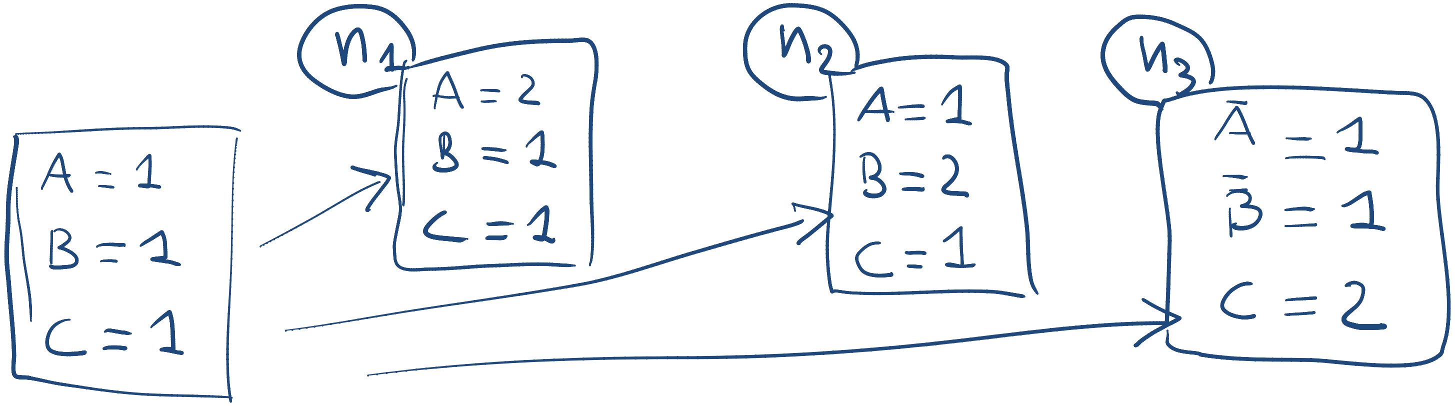
So far we have assumed that we just want to find a possible world that satisfies all the constraints.

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

n_1: $(1+2)+2$ n_2: $(1+1)+1$ n_3: $(2+1)+2$

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

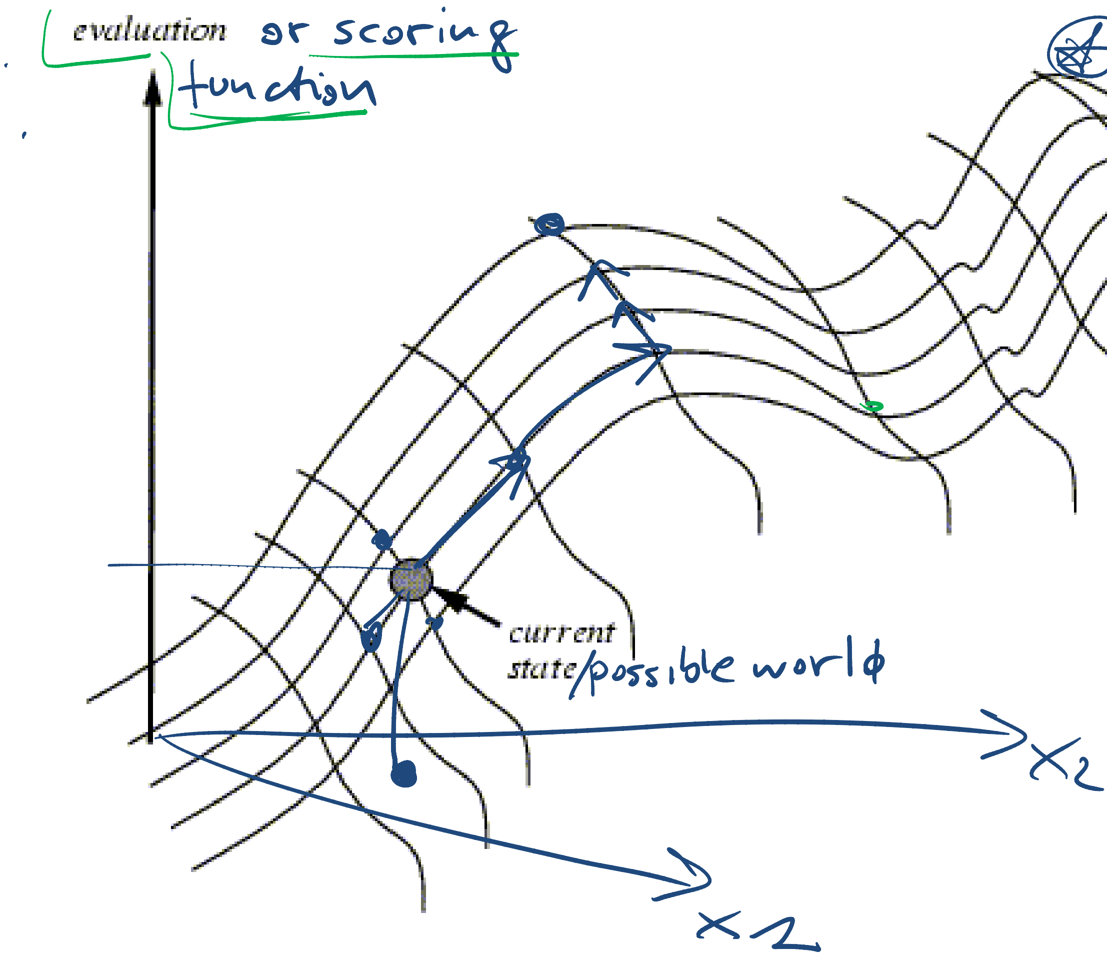
Greedy Descent means selecting the neighbor which minimizes a (cost-based) scoring function. cost + # of conflicts

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

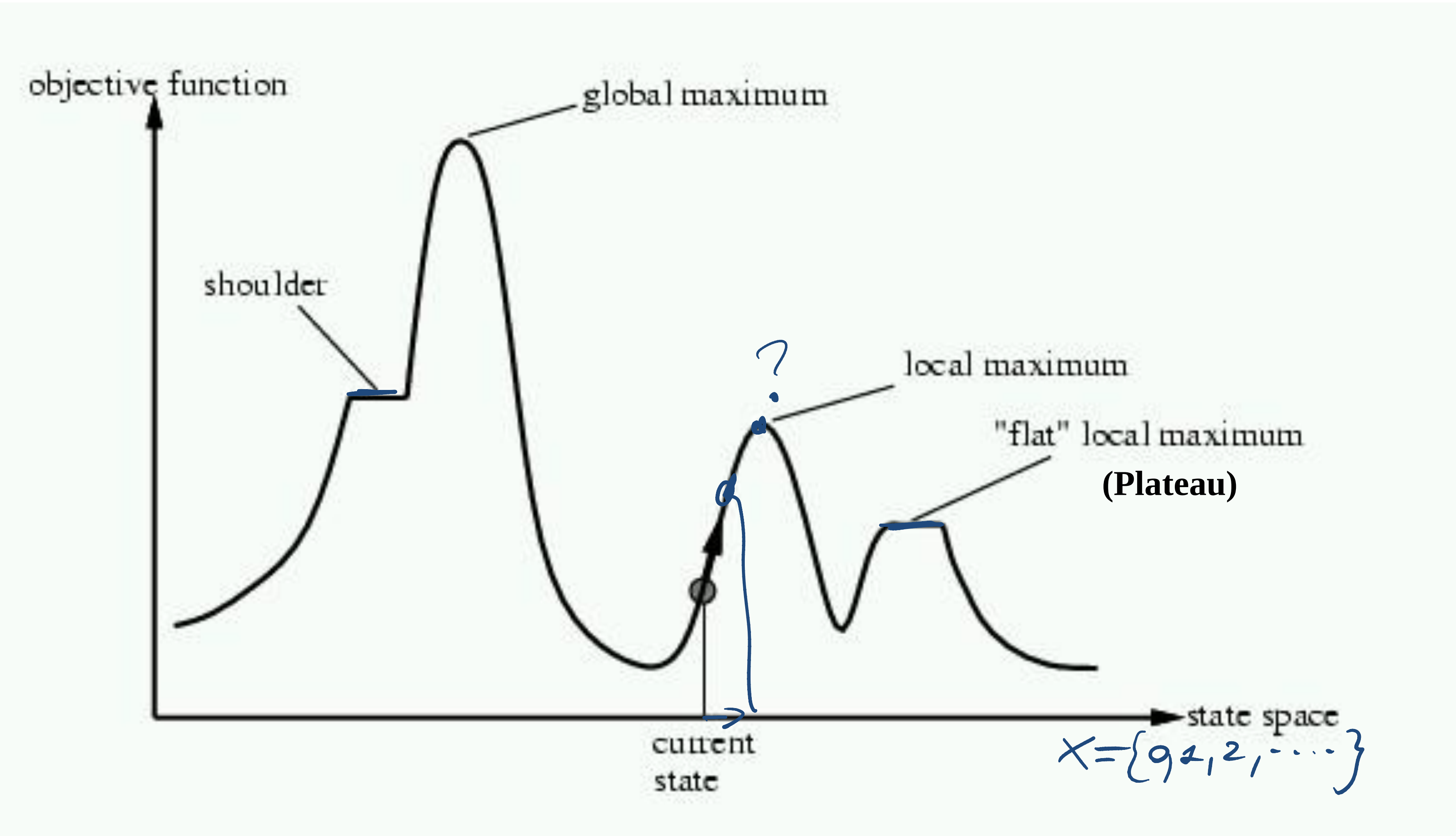


→ two vars
X_1 X_2
assume domain
integer [0, 1, 2 ....]

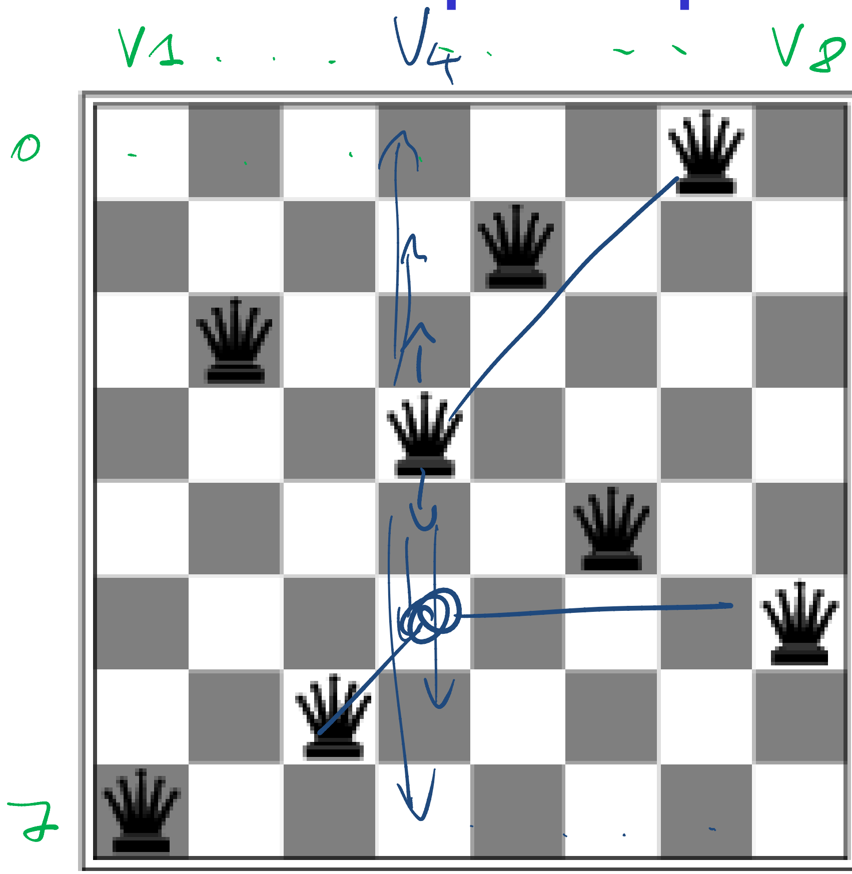
Problems with Hill Climbing

Local Maxima.

Plateau - Shoulders



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



for all the moves (neighbors)

h > 1

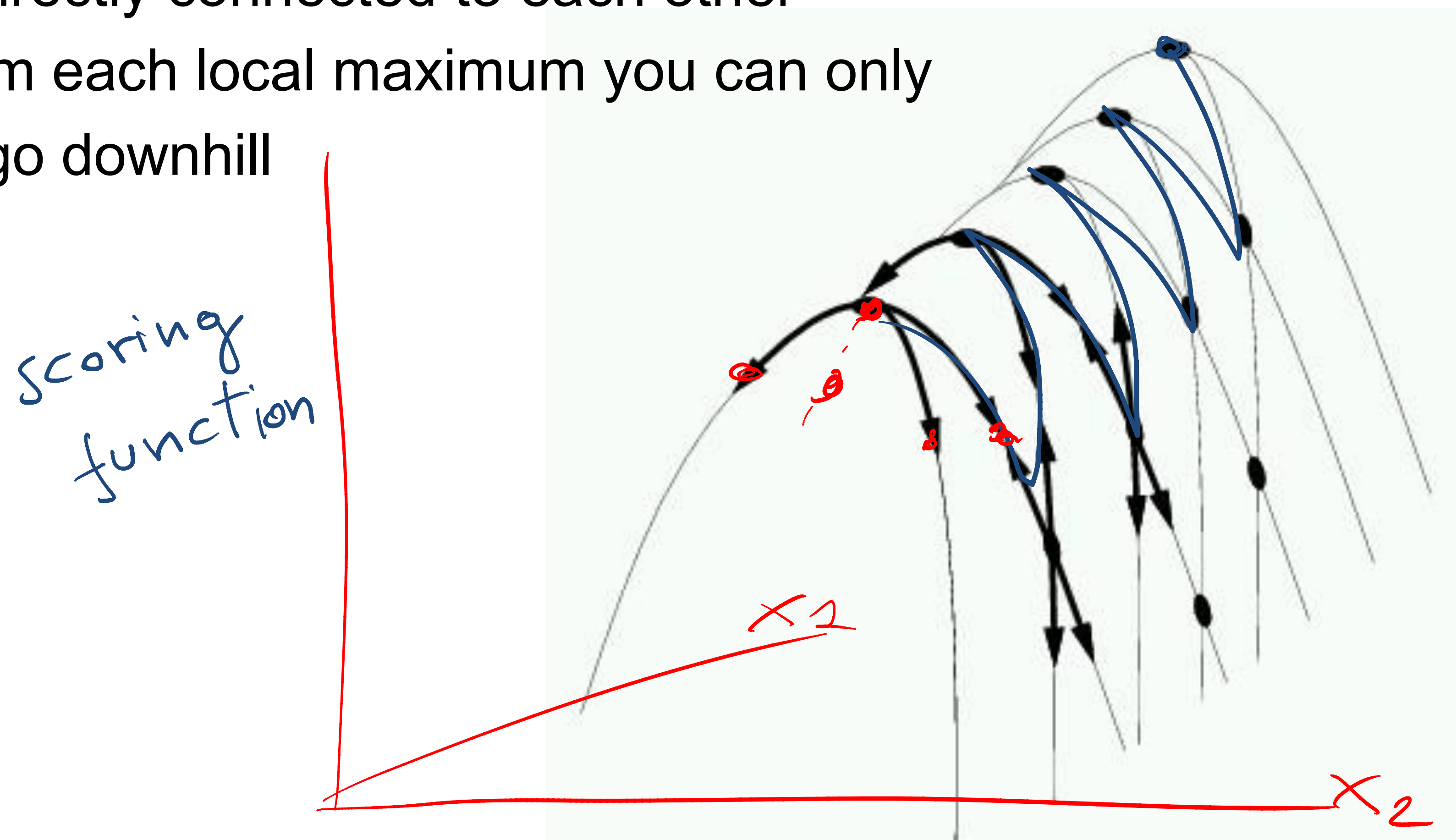
h = 0 for solution

A local minimum with *h = 1*

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



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)