

Constraint Satisfaction Problems

Introduction to Artificial Intelligence

CSE 150 Lecture 4

April 12, 2007

Last lecture

- A* search
- Optimality of A*
- IDA*
- Recursive Best First Search (briefly!)
- Beam Search (really briefly!)

This lecture

- General search applied to CSPs
- Backtracking
- Forward checking
- Heuristics for CSPs

Reading

- Chapter 5 (Meaning, you should have read 1-4 by now!)

Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs)

- A state-space search problem where
- The state is defined by **n variables V_i** ($i=1, \dots, n$)
- The possible values for each variable are from a **domain D_i**
- There are a set of **constraints** between the variable values
- The goal test checks that all variables have been assigned and no constraints are violated.
- We will restrict ourselves to examples with discrete, finite sets of values for each variable.

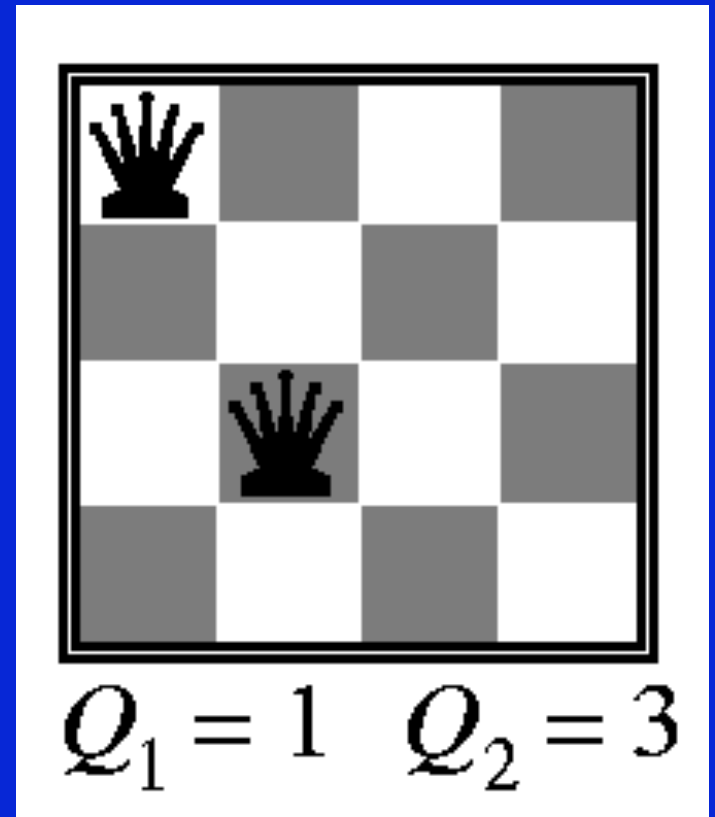
“Toy” Examples

- Sudoku
- N-Queens: Place N queens on an $N \times N$ board such that none can take any other
- Cryptarithmic problems: SEND + MORE = MONEY
- Map coloring (no two neighboring countries the same color)
- Propositional satisfiability: assign T or F to a set of Boolean variables so that a sentence is True (e.g., 3-SAT)

Example: 4-Queens as a CSP

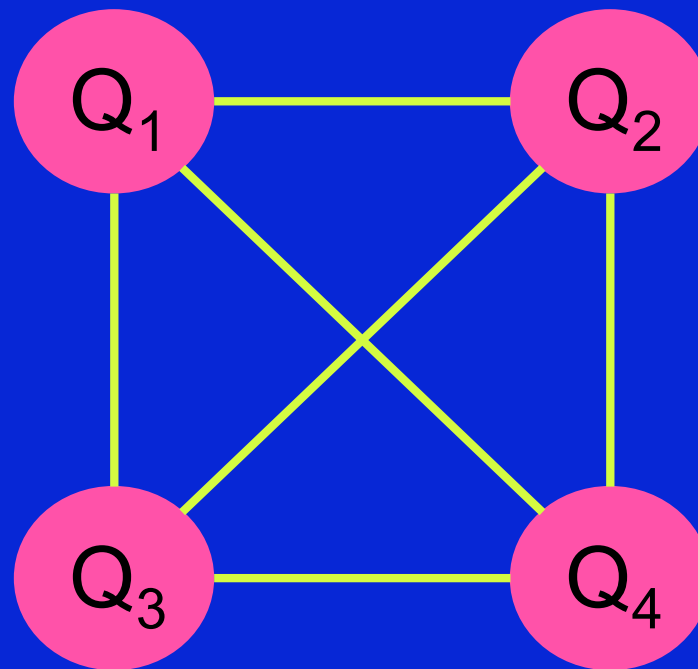
- Place 4 queens on a 4 by 4 chess board such that no queen can take another
- Assume one queen in each column.
Which row does each one go in?
- Variables: Q_i (i is the column)
- Domains: $D_i = \{1, 2, 3, 4\}$ (the row)
- Constraints
 - $Q_i \neq Q_j$ (cannot be in same row)
 - $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)
- Translate each constraint into set of allowable values for its variables

E.g., values for (Q_1, Q_2) are $(1,3)$ $(1,4)$ $(2,4)$ $(3,1)$ $(4,1)$ $(4,2)$



Constraint graph

- **Binary CSP:** each constraint relates at most two variables
- **Constraint graph:** nodes are variables, arcs show constraints



Example: Cryptarithmic

- Variables

D, E, M, N, O, R, S, Y

- Domains

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

- Constraints

$M \neq 0, S \neq 0$ (unary constraints)

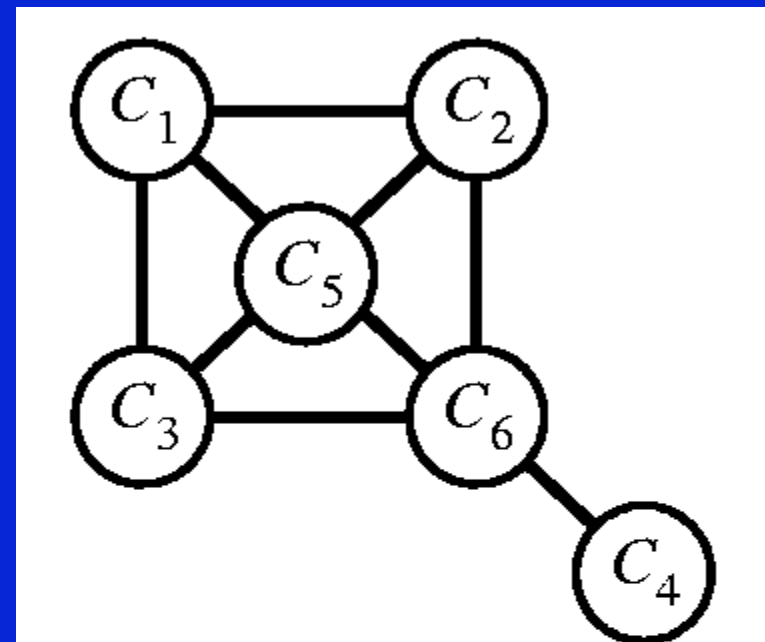
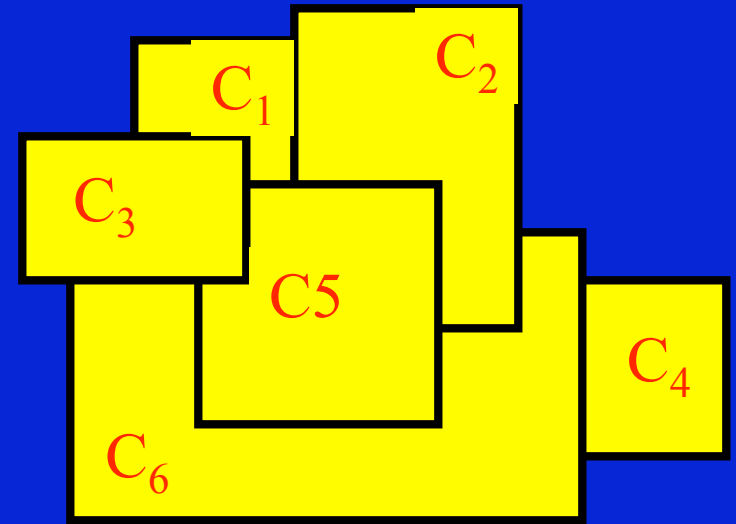
$Y = D + E$ OR $Y = D + E - 10$.

$D \neq E, D \neq M, D \neq N$, etc. (ALL-DIFFERENT)

$$\begin{array}{r} \text{S E N D} \\ + \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

Example: Map coloring

- Color a map so that no adjacent countries have the same color
- Variables:
Countries C_i
- Domains:
{Red, Blue, Green}
- Constraints:
 $C_1 \neq C_2$, $C_1 \neq C_5$, etc.
- Constraint graph:



Real-world CSPs

- Assignment problems
 - e.g., who teaches what class
- Timetabling problems
 - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Planning Observations for the Hubble Space Telescope

- Note that many real-world problems involve real-valued variables as well as binary and discrete variables.

Applying standard search

Let's start with the straightforward, dumb approach, and then fix it

- Initial state: all variables unassigned
 - Operators: assign one of value in domain to one of the unassigned variables.
 - Goal test: all variables assigned, no constraints violated
-
- Notice that this is the same for all CSPs...

Implementation

- CSP state keeps track of which variables have values so far
- Each variable has a domain and a current value

Datatype CSP-STATE

components

UNASSIGNED: a list of variables not yet assigned

ASSIGNED: a list of variables that have values

Datatype CSP-VARIABLE

components

NAME, for I/O purposes

DOMAIN: a list of possible values

VALUE, current value (if any)

- Constraints can be represented
 - explicitly as sets of allowable values, or
 - implicitly by a function that tests for satisfaction of the constraint

Depth-first search applied to 4-Queens

- Start: No variables assigned

- Level 1:

- Q1 = 1 Q1 = 2 ... Q4=3 Q4=4

- Unassigned: Q2-Q4 Un: Q2-Q4 ... Un: Q1-Q3 Un: Q1-Q3

- Level 2:

- Q1=1, Q2=1 Q1=1, Q2=2 ...

- Un: Q3,Q4 Un: Q3,Q4 ...

- This algorithm is called *generate-and-test*, because each state is fully instantiated before it is tested.

Complexity of the Dumb Approach

- Max. depth of space = Depth of solution state:
 $d = n$ (all vars assigned)
- Search algorithm to use:
Depth-first
- Branching factor
 $b = \sum |D_i|$ (at top of tree)
- Time cost
 $O(b^n)$
- How can we do better?

How to not be so Dumb!

- Note Branching factor

$$b = \sum |D_i| \text{ (at top of tree)}$$

- But the assignment of row 1 to Q_1 and row 3 to Q_2 is equivalent first assigning row 3 to Q_2 and then row 1 to Q_1 - the *order of assignment doesn't matter*.
- Hence the idea of *backtracking* search
- Second: We generated partial assignments that violated the constraints! Should check as we generate!

Backtracking search

Use depth-first search, but

1. Fix the order of assignment
 - Branching factor, $b = |D_i|$
2. Check for constraint violations after each assignment
3. **Backtrack** if constraints have been violated

The constraint violation check can be implemented in two ways:

1. Modify Successors function to assign only values that are allowed, given the values already assigned
2. Check constraints are satisfied before expanding a state

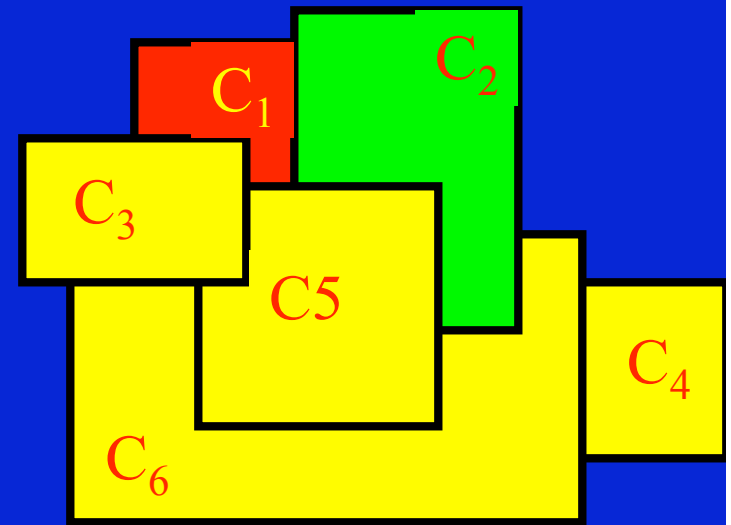
Backtracking search is the basic uninformed algorithm for CSPs

- **Can solve n-queens for $n \approx 15$**

Heuristics for CSPs

More intelligent decisions on:

1. which variable to assign next
2. which value to choose for each variable



- Given $C_1 = \text{RED}$, $C_2 = \text{GREEN}$, which variable to choose next?
 - C_5 : it's the one with the **minimum remaining values (MRV)**
- Given $C_1 = \text{RED}$, $C_2 = \text{GREEN}$, what's a good color for C_3 ?
 - $C_3 = \text{Green}$: **least constraining value**, I.e. Green doesn't constrain the choices for C_5 , but blue would
- Can solve n-queens for $n \approx 1,000$.

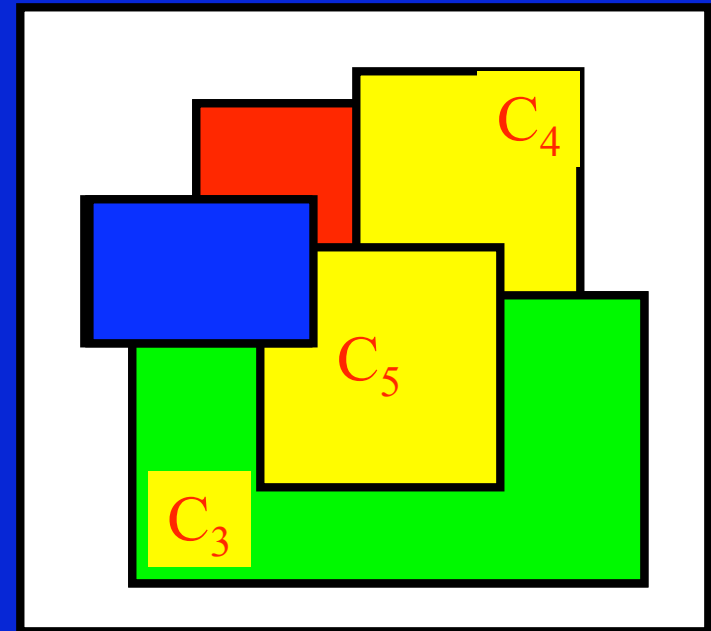
What more can we do?

- Checking for constraint violations as we generate is good - but we can do more:
- Constraint propagation is the idea of propagating the constraints to variables we haven't checked yet.
- The simplest form is ***forward checking***:
 1. Whenever a variable X is assigned, check all variables Y connected to X by a constraint and delete from Y 's domain any value that is inconsistent with the value chosen for X .
 2. As soon as the domain of ***any*** variable becomes empty, backtrack.

Forward checking

- Another way of saying it:
 - Delete illegal values for unassigned variables - *but only the ones connected to the most recent assignment*
 - Backtrack when any variable has no legal values
- Simplified map-coloring example

	RED	BLUE	GREEN
C_1	O		
C_2	X	O	
C_3		X	O
C_4	X		X
C_5	X	X	X

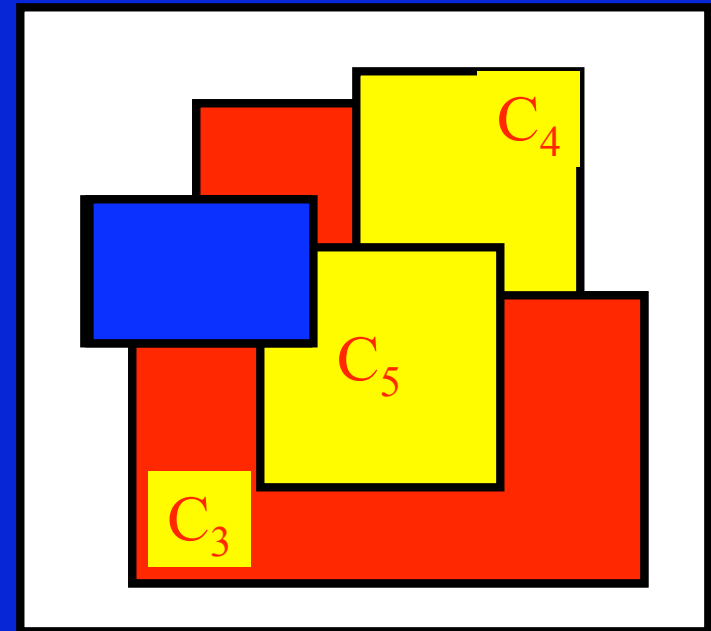


- Can solve n-queens for $n \approx 30$.

Forward checking

- Another way of saying it:
 - Delete illegal values for unassigned variables - *but only the ones connected to the most recent assignment*
 - Backtrack when any variable has no legal values
- Simplified map-coloring example

	RED	BLUE	GREEN
C ₁	O		
C ₂	X	O	
C ₃	O	X	
C ₄	X		
C ₅	X	X	

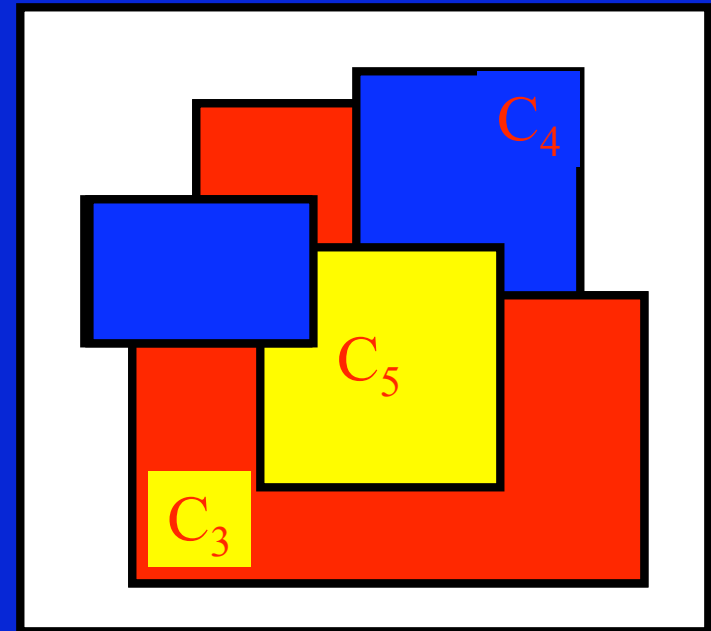


- Can solve n-queens for $n \approx 30$.

Forward checking

- Another way of saying it:
 - Delete illegal values for unassigned variables - *but only the ones connected to the most recent assignment*
 - Backtrack when any variable has no legal values
- Simplified map-coloring example

	RED	BLUE	GREEN
C ₁	O		
C ₂	X	O	
C ₃	O	X	
C ₄	X	O	
C ₅	X	X	

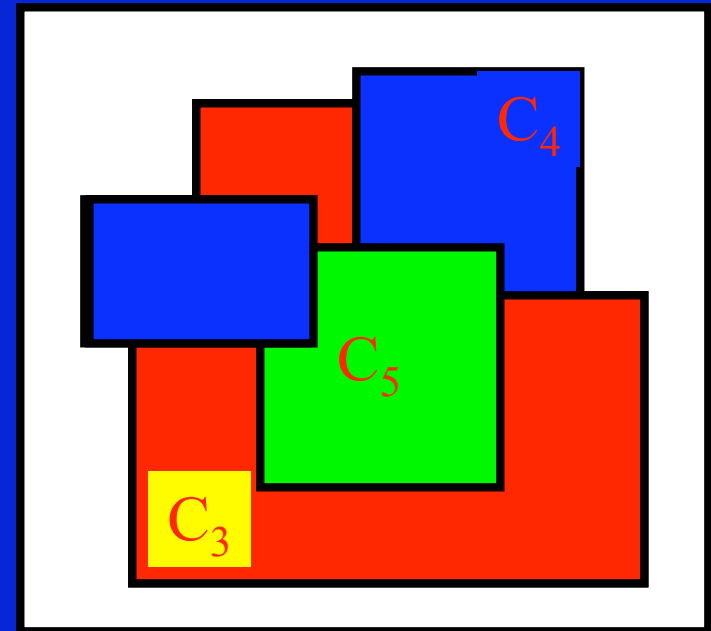


- Can solve n-queens for $n \approx 30$.

Forward checking

- Another way of saying it:
 - Delete illegal values for unassigned variables - *but only the ones connected to the most recent assignment*
 - Backtrack when any variable has no legal values
- Simplified map-coloring example

	RED	BLUE	GREEN
C ₁	O		
C ₂	X	O	
C ₃	O	X	
C ₄	X	O	
C ₅	X	X	O



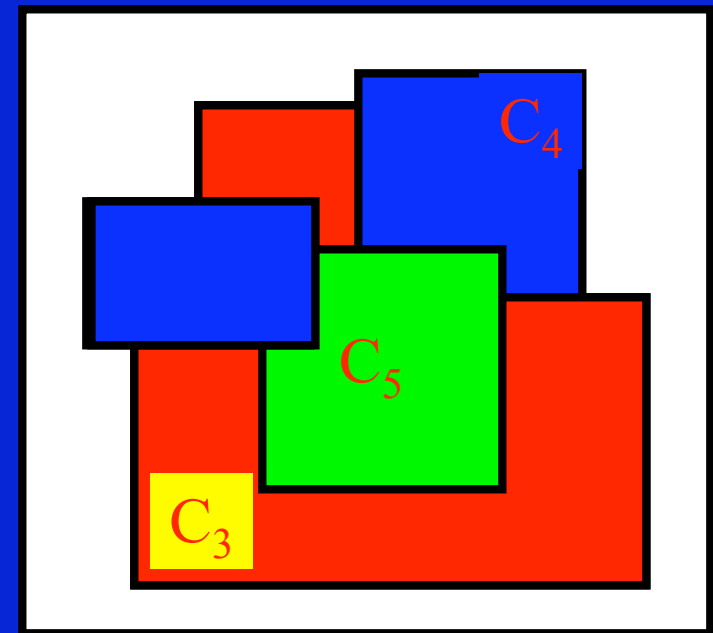
- Can solve n-queens for $n \approx 30$.

Note: Forward checking + MRV

- The obvious partner for Forward checking is the Most Restricted Value (aka Most Constrained Variable) heuristic
- Note that Forward checking is actually helping compute what MRV needs to do its job!

	RED	BLUE	GREEN
C ₁	O		
C ₂	X	O	
C ₃	O	X	
C ₄	X	O	
C ₅	X	X	O

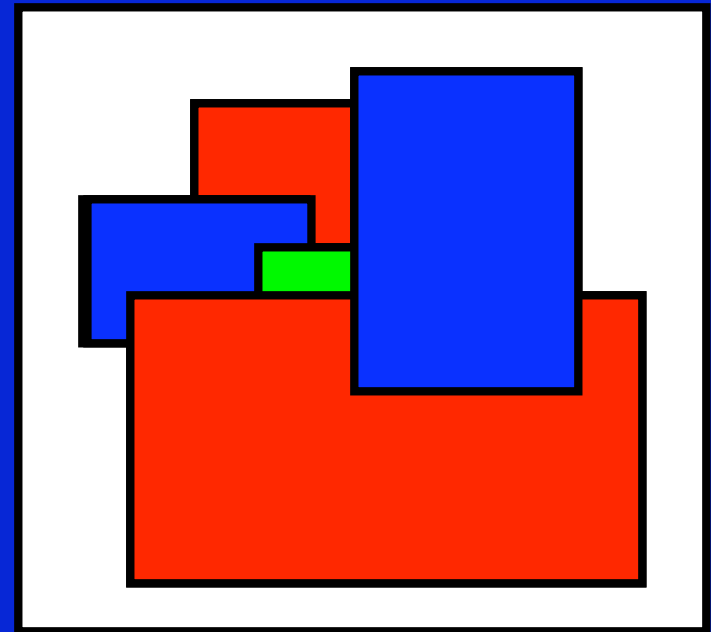
• Can solve n-queens for $n \approx 30$.



Constraint Propagation

- Forward checking on steroids:
- Start with forward checking, but *continue* checking any variables connected to variables you have deleted values for - until no more values have been deleted.
- The algorithm for this is called *arc-consistency* (see *book!*)

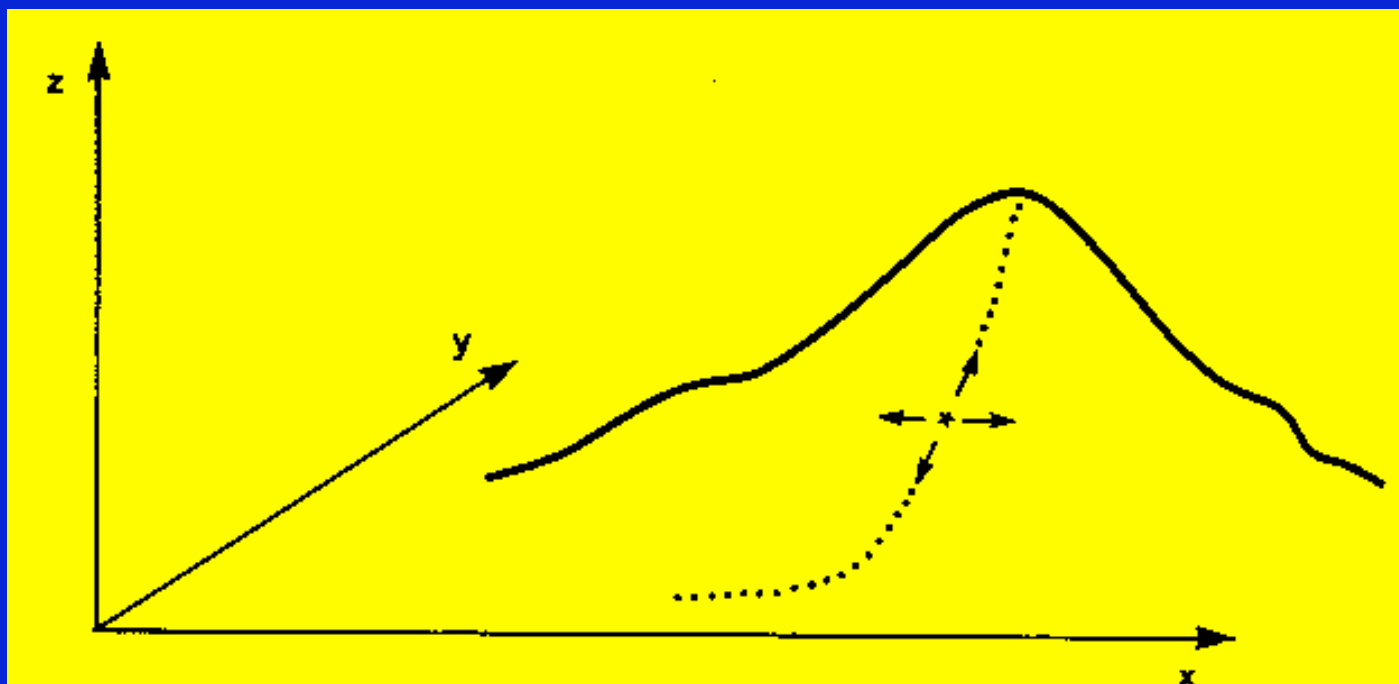
	RED	BLUE	GREEN
C ₁	O		
C ₂	X	O	
C ₃	O	X	X
C ₄	X	O	X
C ₅	X	X	O



- *Note: Last three steps were only constraint propagation!*

Hill-climbing (or gradient ascent/descent)

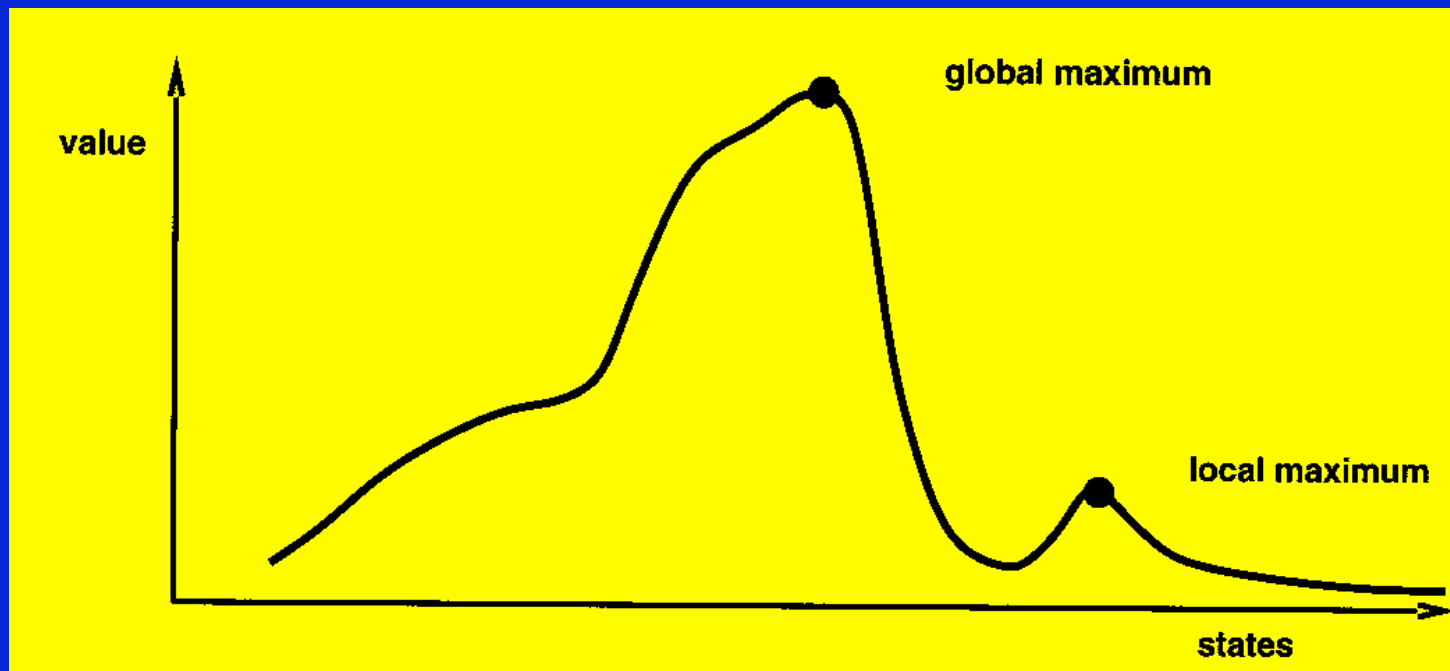
- Define quality function on state space $h(n)$
- Make local changes of state to increase value of function.



- ``Like climbing Everest in thick fog with amnesia''

Hill Climbing Continued

- Depending upon initial conditions, can get stuck in local maxima



- Plateaus lead to random walk around state space
- Simulated annealing

Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with “complete” states, i.e., *all variables assigned*

To apply to CSPs:

- allow states with violated constraints
- operators reassign variable values
- Reducing constraints

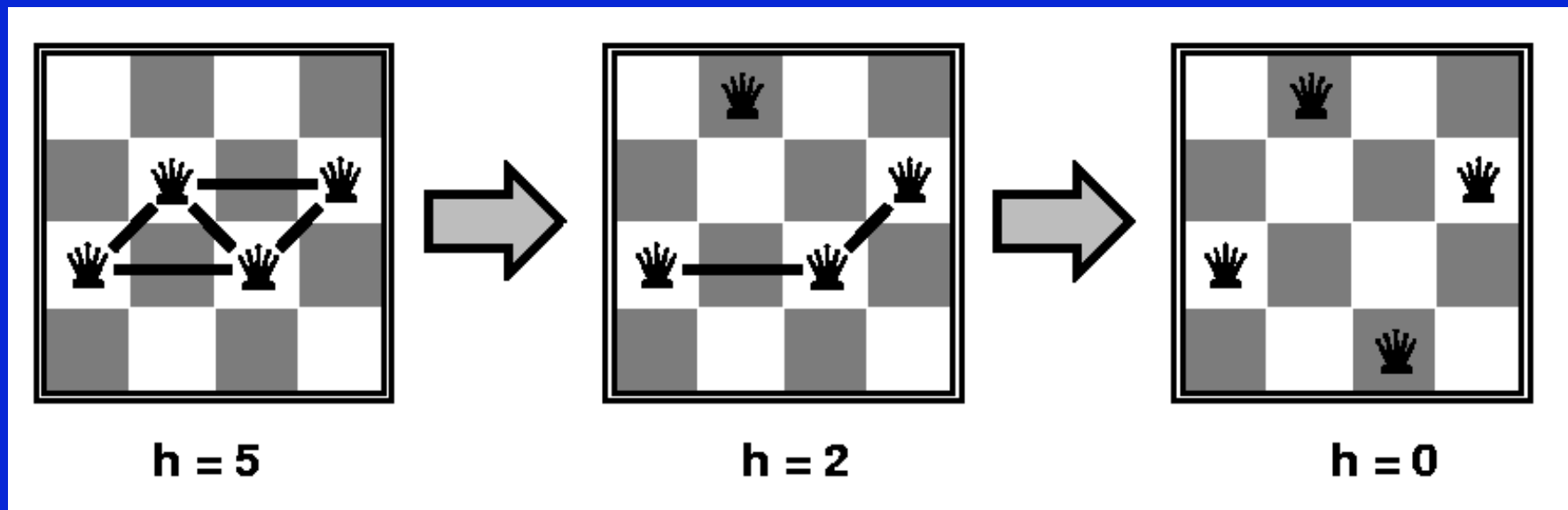
Variable selection: randomly select any conflicted variable

min-conflicts heuristic:

- choose value that violates the fewest constraints
- i.e., hill-descend with $h(n) = \text{total number of violated constraints}$

Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: Move queen in column
- Goal test: No attacks
- Evaluation: $h(n) = \text{number of attacks}$



- Can solve n-queens with high probability ($n=10,000,000$)

Summary

- CSPs are a special kind of problem:
 - States defined by values of a fixed set of variables
 - Goal test defined by constraints on variable values
- Backtracking = depth-first search with
 - 1) fixed variable order
 - 2) only legal successors
- Forward checking prevents assignments that guarantee later failure
- Variable ordering and value selection heuristics help significantly
- Iterative min-conflicts is usually effective in practice