# Use and Distribution of these Slides

- These slides are primarily intended for the students in classes I teach. In some cases, I only make PDF versions publicly available. If you would like to get a copy of the originals (Apple KeyNote or Microsoft PowerPoint), please contact me via email at fkurfess@calpoly.edu. I hereby grant permission to use them in educational settings. If you do so, it would be nice to send me an email about it. If you're considering using them in a commercial environment, please contact me first.

# Logistics

- **Senior Project working with Emotiv (Adam Rizkalla) on using BCI devices for adaptive music playlists**
- **AI Nugget presentations**
  - proposals and past presentations mostly graded
  - Section 1:
    - Stephen Calabrese: Wolfram Alpha
    - Brandon Page: Google Now
    - Adin Miller: AI System Builds Video Games
  - Section 3:
    - Luke Larson: Crusher
    - Jorge Mendoza: Behind IBM’s Watson
- **Bot/WumpusEnvironment**
  - source code, JavaDocs for WumpusEnvironment is on PolyLearn
  - post insights of potential interest for others on the PolyLearn forum

# Chapter Overview
# Search

- Motivation
- Objectives
- Search as Problem-Solving
  - problem formulation
  - problem types
- Uninformed Search
  - breadth-first
  - depth-first
  - uniform-cost search
  - depth-limited search
  - iterative deepening
  - bi-directional search
- Informed Search
  - best-first search
  - search with heuristics
  - memory-bounded search
  - iterative improvement search
- Non-Traditional Search
  - local search and optimization
  - constraint satisfaction
  - search in continuous spaces
  - partially observable worlds
- Important Concepts and Terms
- Chapter Summary

# Revisions:

- 2009-10-14
  - split this chapter into two parts, to prepare for the transition to the 3rd edition
  - added some pictures and diagrams
- This set of slides is the second one of the part on search algorithms.

# Motivation

- "conventional" search strategies and the respective algorithms are becoming more and more part of the "standard" computer science area
- some alternative search methods have been developed that deal with problems and domains for which the conventional ones are not very well suited
  - some of these methods originated outside of the search-based approaches, but can be viewed as search methods

# Objectives

- identify applications and tasks where search in general is a suitable approach, but conventional search methods have serious drawbacks
- be familiar with some of the approaches to non-conventional search
    - local search and optimization
    - constraint satisfaction
    - search in continuous spaces, partially observable worlds
- evaluate the suitability of a search strategy for a problem
    - completeness, time & space complexity, optimality
    - dealing with memory limitations, partial observability, non-deterministic outcomes of actions, continuous search spaces, and unknown environments

# Non-Traditional Search

- local search and optimization
- constraint satisfaction
- search in continuous spaces
- partially observable worlds

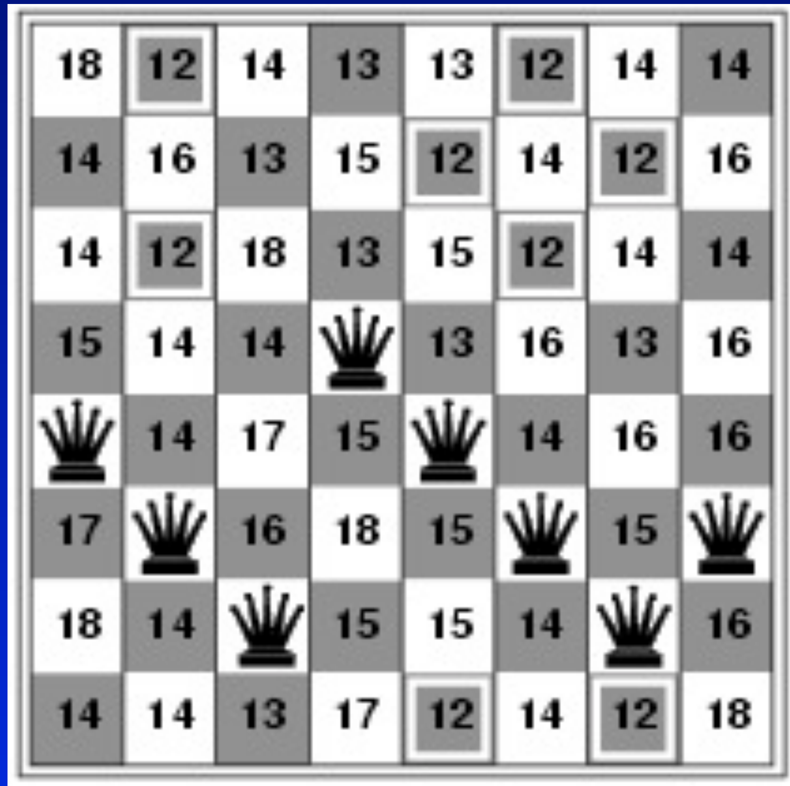
# Local Search and Optimization

- for some problem classes, it is sufficient to find a solution
  - the path to the solution is not relevant
- memory requirements can be dramatically relaxed by modifying the current state
  - all previous states can be discarded
  - since only information about the current state is kept, such methods are called *local*

# Example: *n*-queens

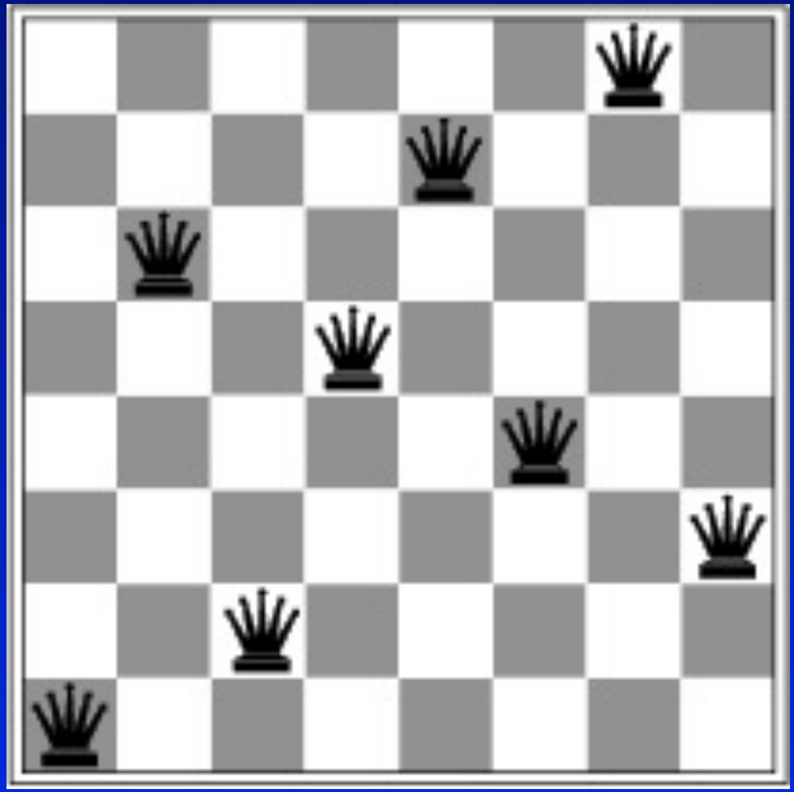
- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal

# Hill-climbing search: 8-queens problem




- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

# Hill-climbing search: 8-queens problem



- A local minimum with *h = 1*

# Iterative Improvement Search

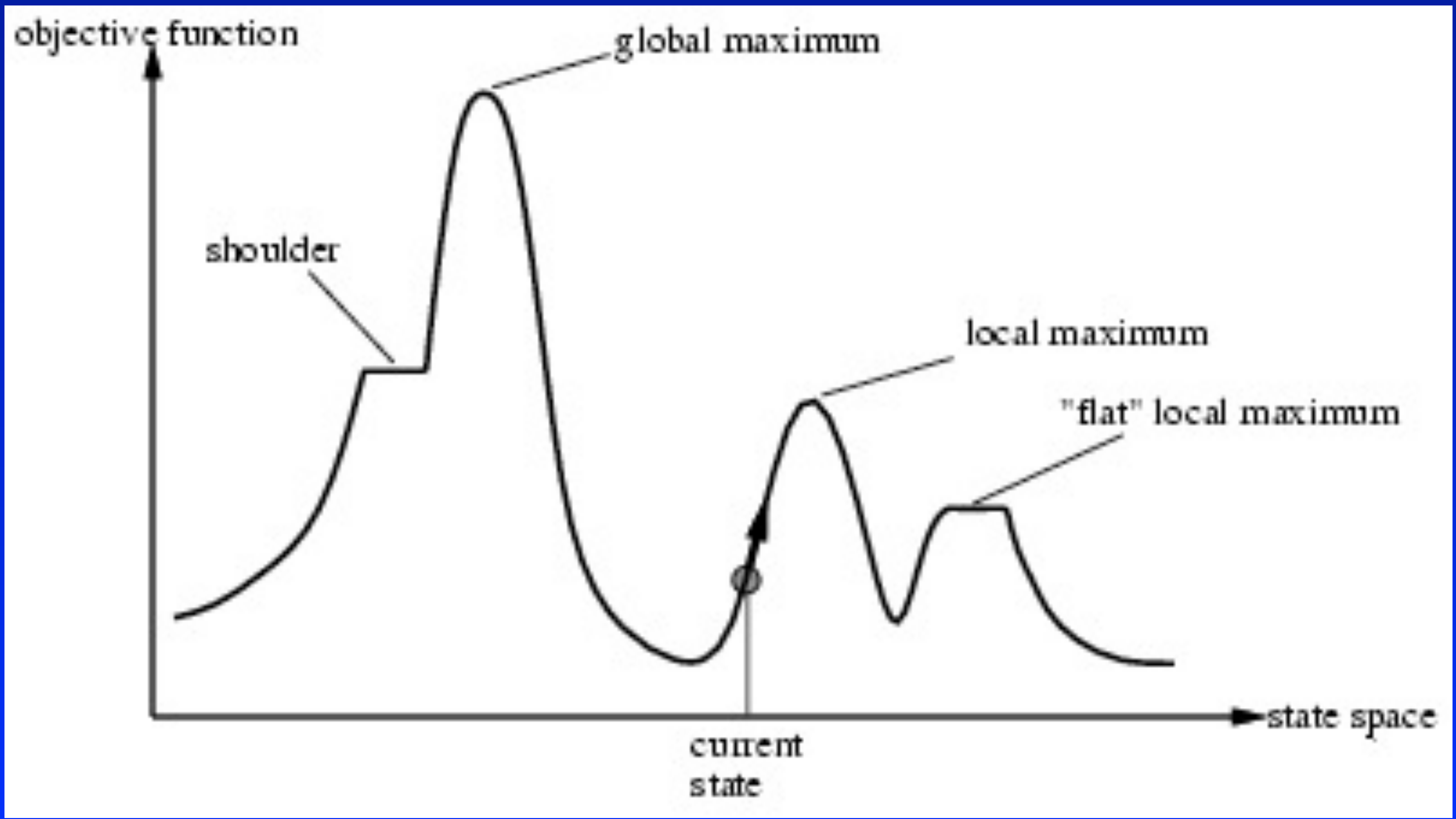
- for some problems, the state description provides all the information required for a solution
  - path costs become irrelevant
  - global maximum or minimum corresponds to the optimal solution
- iterative improvement algorithms start with some configuration, and try modifications to improve the quality
  - 8-queens: number of un-attacked queens
  - VLSI layout: total wire length
- analogy: state space as landscape with hills and valleys

# Hill-Climbing Search

- continually moves uphill
  - increasing value of the evaluation function
  - gradient descent search is a variation that moves downhill
- very simple strategy with low space requirements
  - stores only the state and its evaluation, no search tree
- problems
  - local maxima
    - algorithm can’t go higher, but is not at a satisfactory solution
  - plateau
    - area where the evaluation function is flat
  - ridges
    - search may oscillate slowly

# Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



# Simulated Annealing

- similar to hill-climbing, but some down-hill movement
  - random move instead of the best move
  - depends on two parameters
    - $\Delta E$, energy difference between moves; T, temperature
    - temperature is slowly lowered, making bad moves less likely
- analogy to annealing
  - gradual cooling of a liquid until it freezes
- will find the global optimum if the temperature is lowered slowly enough
- applied to routing and scheduling problems
  - VLSI layout, scheduling

# Local Beam Search

- variation of *beam search*
  - a path-based method that looks at several paths “around” the current one
- keeps *k* states in memory, instead of only one
  - information between the states can be shared
    - moves to the most promising areas
- *stochastic* local beam search selects the *k* successor states randomly
  - with a probability determined by the evaluation function

# Logistics - Oct. 16, 2012

- **AI Nugget presentations scheduled for Oct. 11**
  - Section 1:
    - William Budney: SwiftKey
    - Grant Frame: Autonomous Agile Aerial Robots
    - Drew Bentz: Stand Up Comedy Robot
    - Chris Colwell: Watson, IBM's Brainchild
    - stephen calabrese: Wolfram Alpha (carried over from Oct. 11)
    - Brandon Page: Google Now (carried over from Oct. 11)
  - Section 3:
    - Therin Irwin: Intelligent Databases
    - Brian Gomberg: Robot SWARM
    - Bassem Tossoun: Don't Worry, I've Got Siri
- **Assignments and Labs**
  - A1: Search Algorithms
    - deadline Tue, Oct. 23
  - Lab 4: extensions available until Sun, Oct. 21
  - Lab 5 available: *AI in Real Life*
  - Lab submission deadlines fixed: Tue, end of day (not end of lab)
- **Quiz 4**
  - available all day Tue, Oct. 16
- **Project**
  - mid-quarter project fair on Thu, Oct. 25
- **Zynga Event today at 5:30 in 14-252**
  - free food, presentation about getting a job, giveaways

# Genetic Algorithms (GAs)

- variation of stochastic beam search
  - successor states are generated as variations of *two* parent states, not only one
  - corresponds to natural selection with sexual reproduction
  - mutation provides an additional random element
    - random modification of features (variable values)

# GA Terminology

- population
  - set of *k* randomly generated states
- generation
  - population at a point in time
  - usually, propagation is synchronized for the whole population
- individual
  - one element from the population
  - described as a string over a finite alphabet
    - binary, ACGT, letters, digits
    - consistent for the whole population
- fitness function
  - evaluation function in search terminology
  - higher values lead to better chances for reproduction

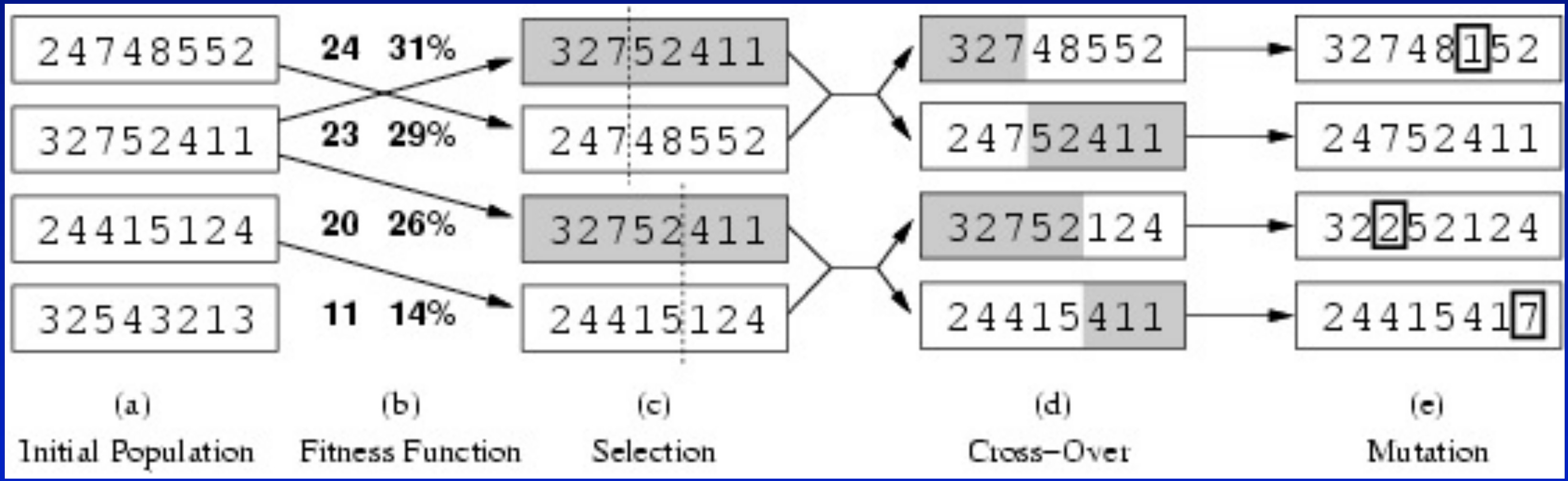
# GA Principles

- reproduction
  - the state description of the two parents is split at the crossover point
    - determined in advance, often randomly chosen
    - must be the same for both parents
  - one part is combined with the other part of the other parent
    - one or both of the descendants may be added to the population
    - compatible state descriptions should assure viable descendants
      - depends on the choice of the representation
      - may not have a high fitness value
- mutation
  - each individual may be subject to random modifications in its state description
    - usually with a low probability
- schema
  - useful components of a solution can be preserved across generations

# GA Applications

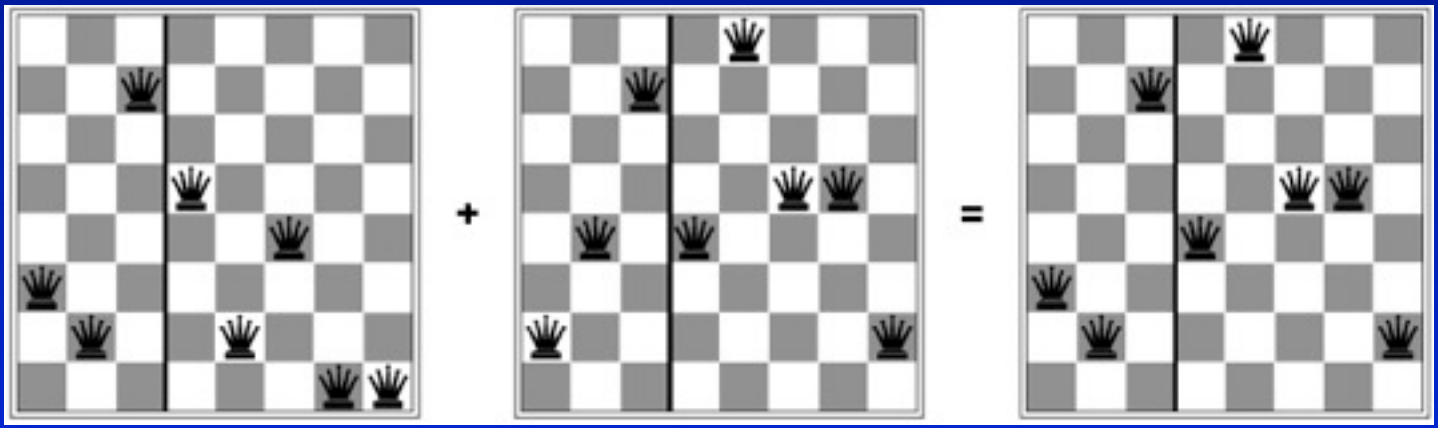
- often used for optimization problems
  - circuit layout, system design, scheduling
- termination
  - “good enough” solution found
  - no significant improvements over several generations
  - time limit

# Example: Genetic algorithm for N-Queens




- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

# Genetic Algorithm for N-Queens



# Constraint Satisfaction

- satisfies additional structural properties of the problem
  - may depend on the representation of the problem
- the problem is defined through a set of domain variables
  - variables can have possible values specified by the problem
  - constraints describe allowable combinations of values for a subset of the variables
- *state* in a CSP
  - defined by an assignment of values to some or all variables
- *solution* to a CSP
  - must assign values to all variables
  - must satisfy all constraints
  - solutions may be ranked according to an objective function

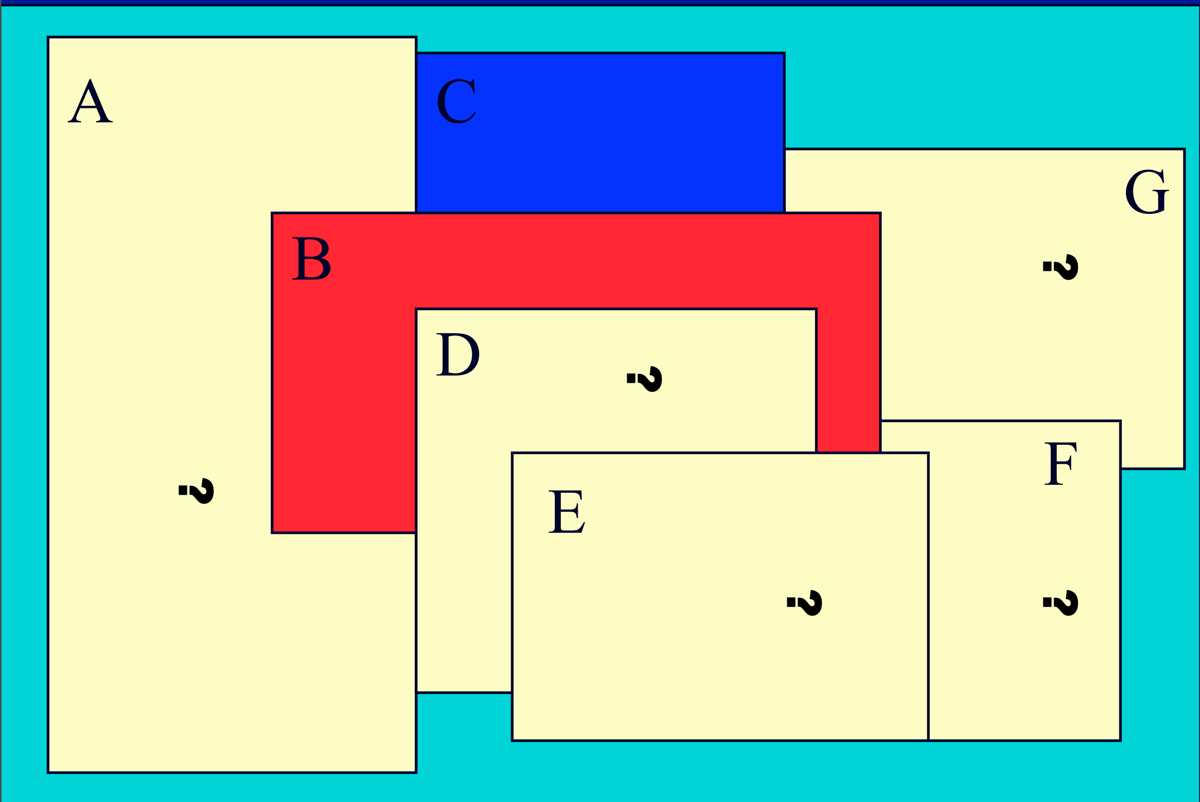
# CSP Approach

- the goal test is decomposed into a set of constraints on variables
  - checks for violation of constraints before new nodes are generated
    - must backtrack if constraints are violated
  - forward-checking looks ahead to detect unsolvability
    - based on the current values of constraint variables

# CSP Example: Map Coloring

- color a map with three colors so that adjacent countries have different colors




variables:
*A, B, C, D, E, F, G*

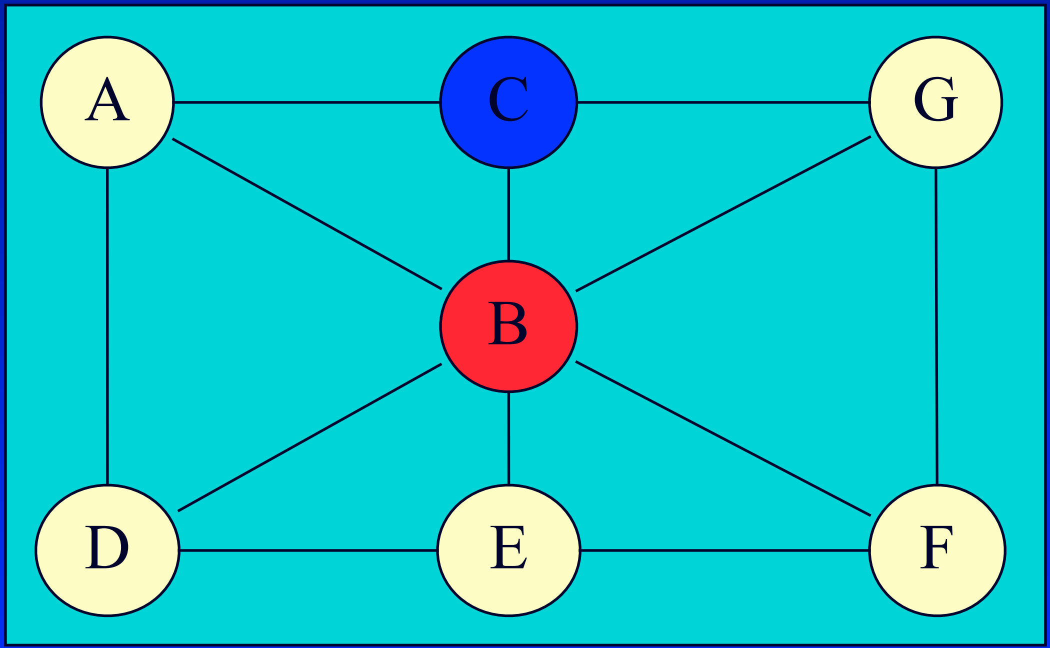
values:
*{red, green, blue}*

constraints:
*"no neighboring regions have the same color"*

legal combinations for A, B:
*{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}*

# Constraint Graph

- visual representation of a CSP
  - variables are nodes
  - arcs are constraints




the map coloring example
represented as constraint graph

# Example: Map-Coloring

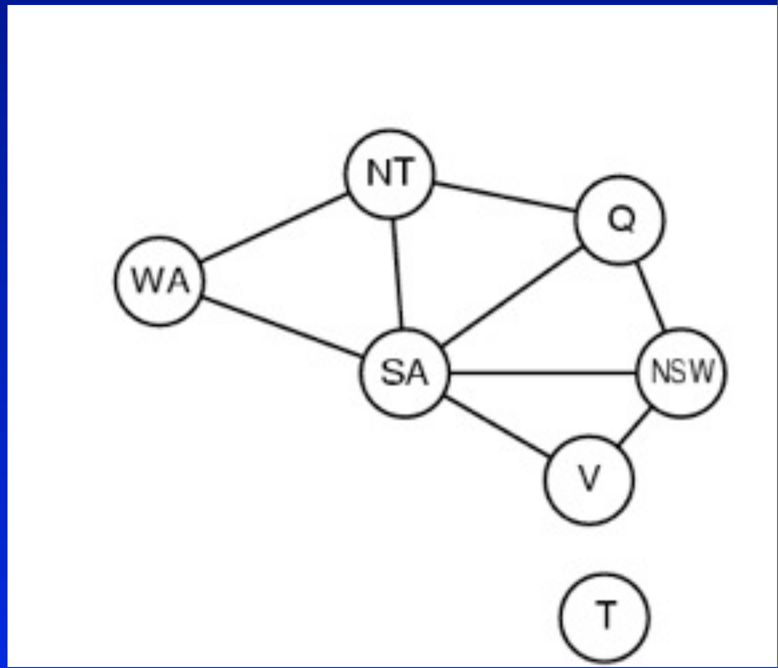

- Variables WA, NT, Q, NSW, V, SA, T
- Domains $D_i$ = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

# Example: Australia Map-Coloring


- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

# Constraint graph

- Binary CSP: each constraint relates two variables
  - arcs connect exactly two nodes
  - n-ary CSP uses hypergraphs where arcs can connect more than two nodes
- Constraint graph: nodes are variables, arcs are constraints

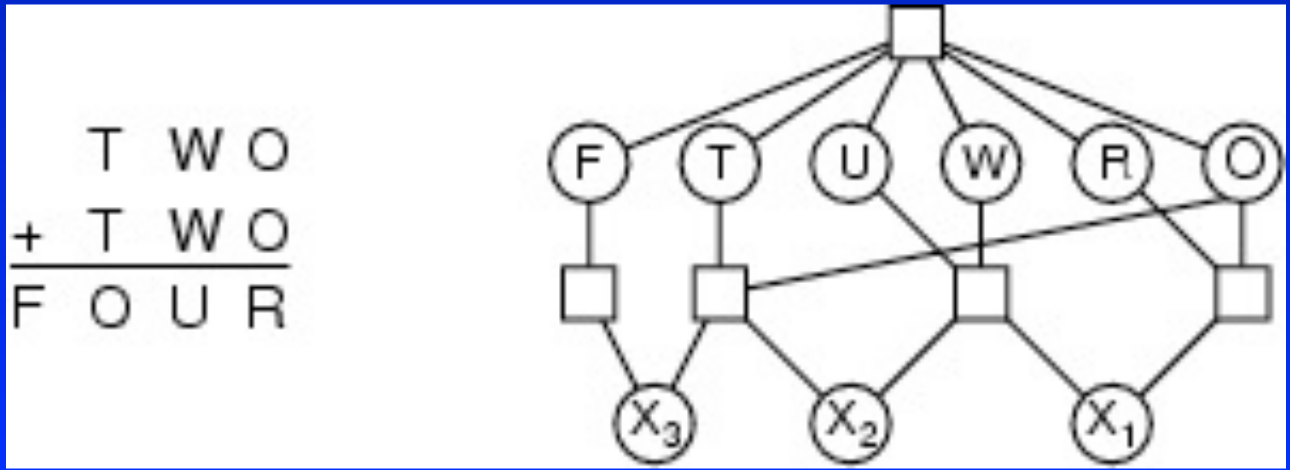


# Varieties of CSPs

- Discrete variables
  - finite domains:
    - n variables, domain size d O(dn) complete assignments
    - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., StartJob1 + 5 ≤ StartJob3
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

# Example: Cryptarithmetic

- Variables: F T U W R O $X_1$ $X_2$ $X_3$
- Domains: {0,1,2,3,4,5,6,7,8,9}
- Constraints: Alldiff (F,T,U,W,R,O)
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F$, $T \neq 0$, $F \neq 0$



# Benefits of CSP

- standardized representation pattern
  - variables with assigned values
  - constraints on the values
  - allows the use of generic heuristics
    - no domain knowledge is required

# CSP as Incremental Search Problem

- initial state
  - all (or at least some) variables unassigned
- successor function
  - assign a value to an unassigned variable
  - must not conflict with previously assigned variables
- goal test
  - all variables have values assigned
  - no conflicts possible
    - not allowed in the successor function
- path cost
  - e.g. a constant for each step
  - may be problem-specific

# CSPs and Search

- in principle, any search algorithm can be used to solve a CSP
  - awful branching factor
    - *n*d* for *n* variables with *d* values at the top level, *(n-1)*d* at the next level, etc.
  - not very efficient, since they neglect some CSP properties
    - commutativity: the order in which values are assigned to variables is irrelevant, since the outcome is the same

# Backtracking Search for CSPs

- a variation of depth-first search that is often used for CSPs
  - values are chosen for one variable at a time
  - if no legal values are left, the algorithm backs up and changes a previous assignment
  - very easy to implement
    - initial state, successor function, goal test are standardized
  - not very efficient
    - can be improved by trying to select more suitable unassigned variables first

# Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

# Heuristics for CSP

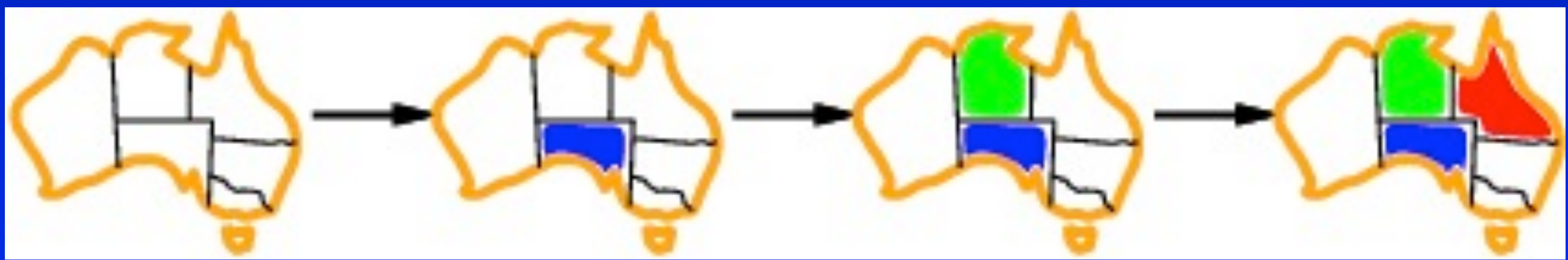
- most-constrained variable (minimum remaining values, “fail-first”)
  - variable with the fewest possible values is selected
  - tends to minimize the branching factor
- most-constraining variable
  - variable with the largest number of constraints on other unassigned variables
- least-constraining value
  - for a selected variable, choose the value that leaves more freedom for future choices

# Analyzing Constraints

- forward checking
  - when a value $X$ is assigned to a variable, inconsistent values are eliminated for all variables connected to $X$
    - identifies “dead” branches of the tree before they are visited
- constraint propagation
  - analyses interdependencies between variable assignments via *arc consistency*
    - an arc between $X$ and $Y$ is consistent if for every possible value $x$ of $X$, there is some value $y$ of $Y$ that is consistent with $x$
    - more powerful than forward checking, but still reasonably efficient
    - but does not reveal every possible inconsistency

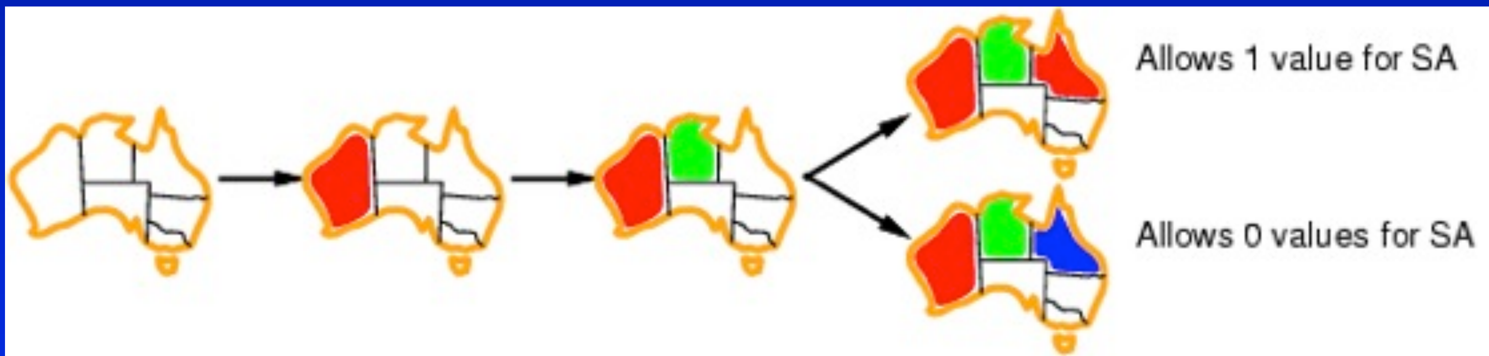
# Most constraining variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



# Least constraining value

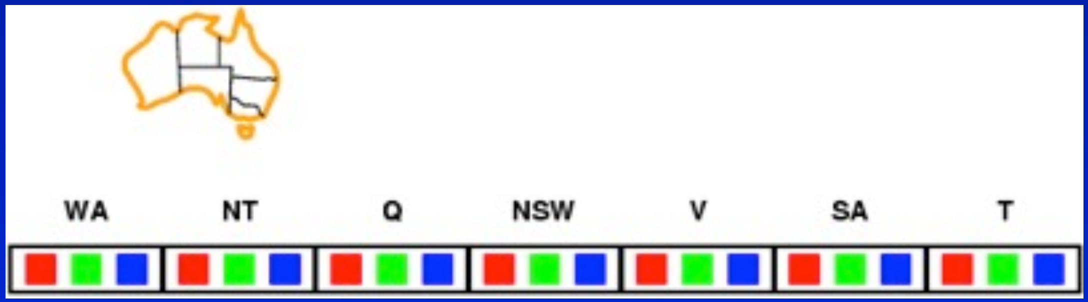
- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables




- Combining these heuristics makes 1000 queens feasible

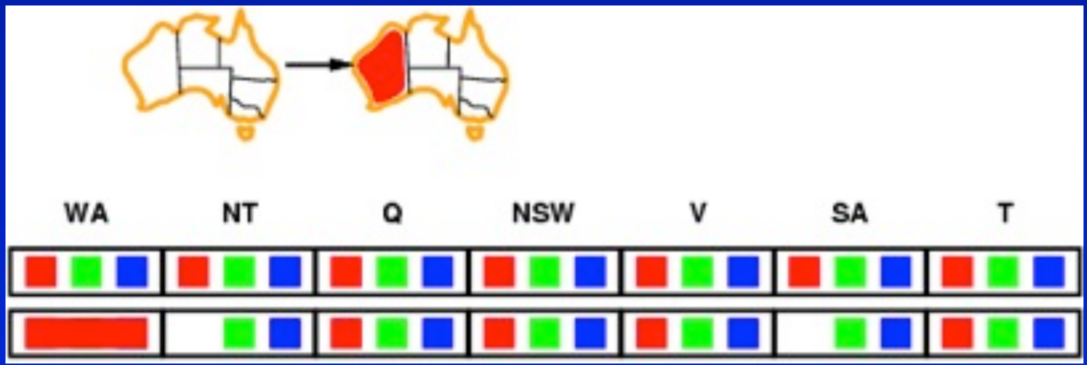
# Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



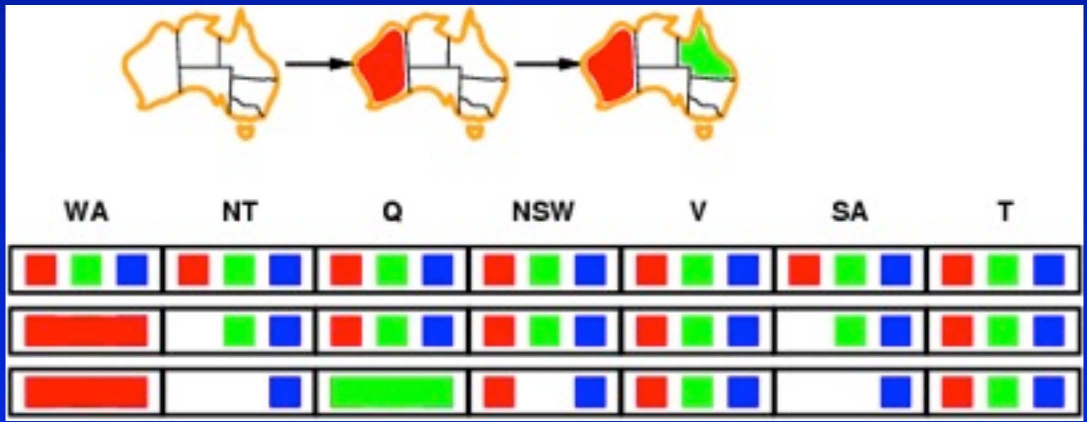
# Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



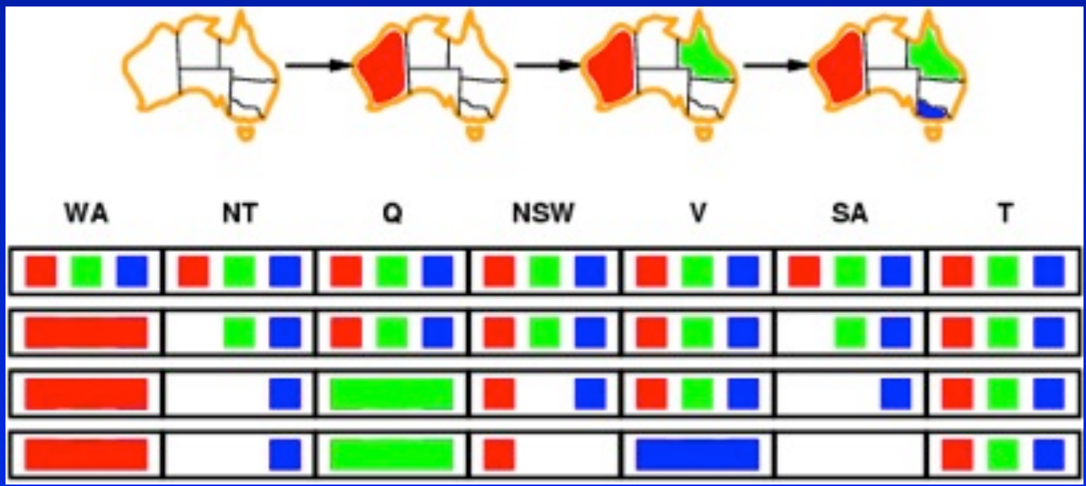
# Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



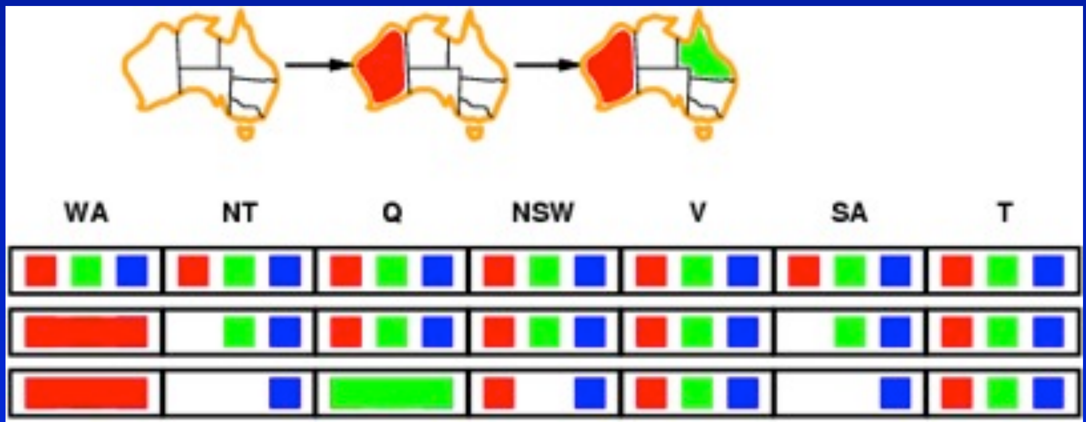
# Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



# Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

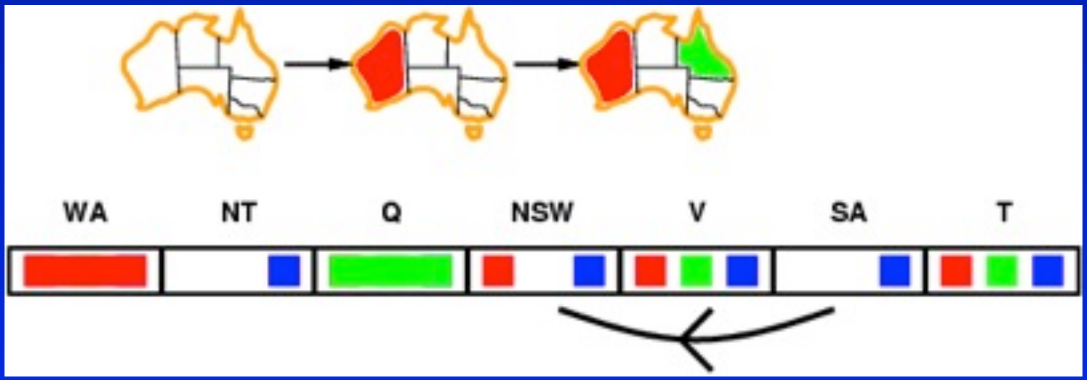



- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

# Arc consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff

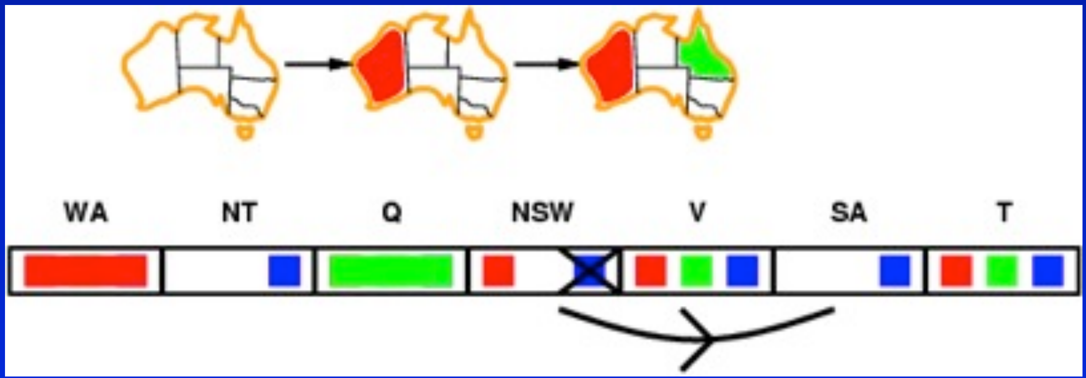
for every value x of X there is some allowed y



# Arc consistency

- Simplest form of propagation makes each arc consistent
- X → Y is consistent iff

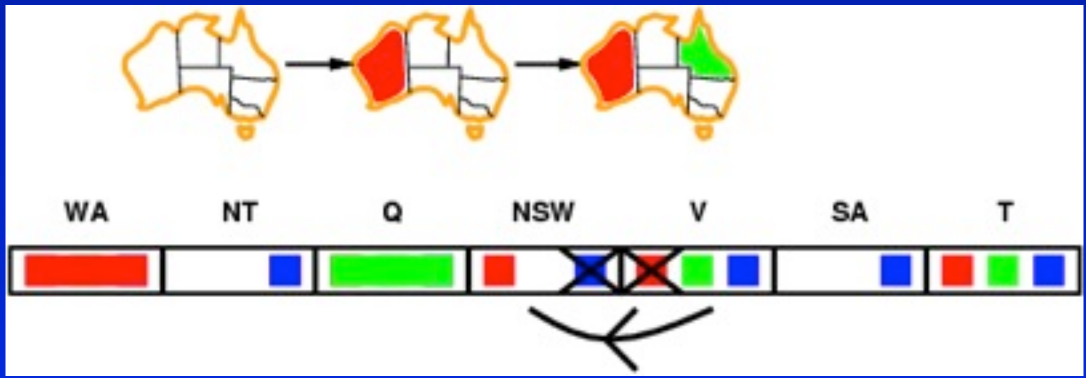
for every value x of X there is some allowed y



# Arc consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff

  for every value x of X there is some allowed y

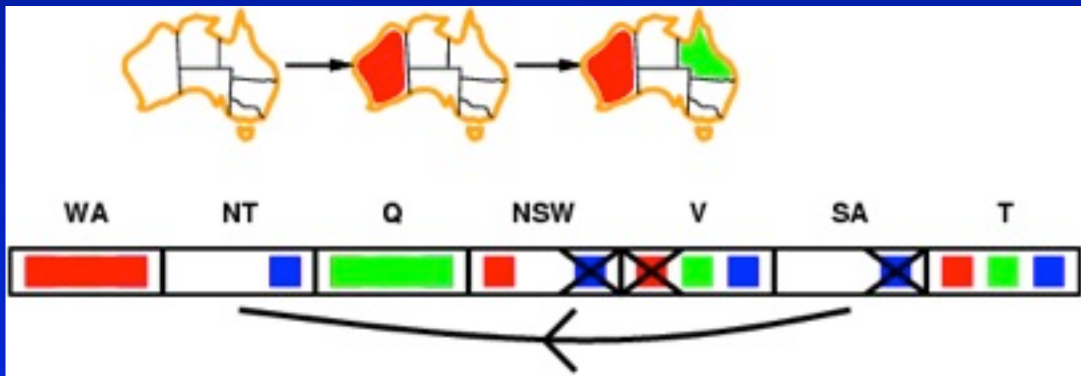



- If X loses a value, neighbors of X need to be rechecked

# Arc consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff

  for every value x of X there is some allowed y




- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

# Local Search and CSP

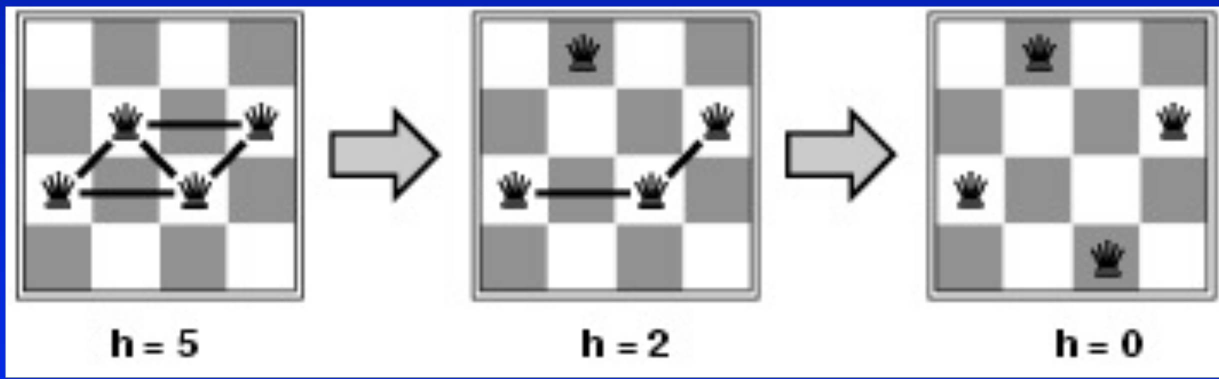
- local search (iterative improvement) is frequently used for constraint satisfaction problems
  - values are assigned to all variables
  - modification operators move the configuration towards a solution
- often called heuristic repair methods
  - repair inconsistencies in the current configuration
- simple strategy: min-conflicts
  - minimizes the number of conflicts with other variables
  - solves many problems very quickly
    - million-queens problem in less than 50 steps
- can be run as *online* algorithm
  - use the current state as new initial state

# Local search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with h(n) = total number of violated constraints

# Example: 4-Queens

- States: 4 queens in 4 columns ($4^4$ = 256 states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks



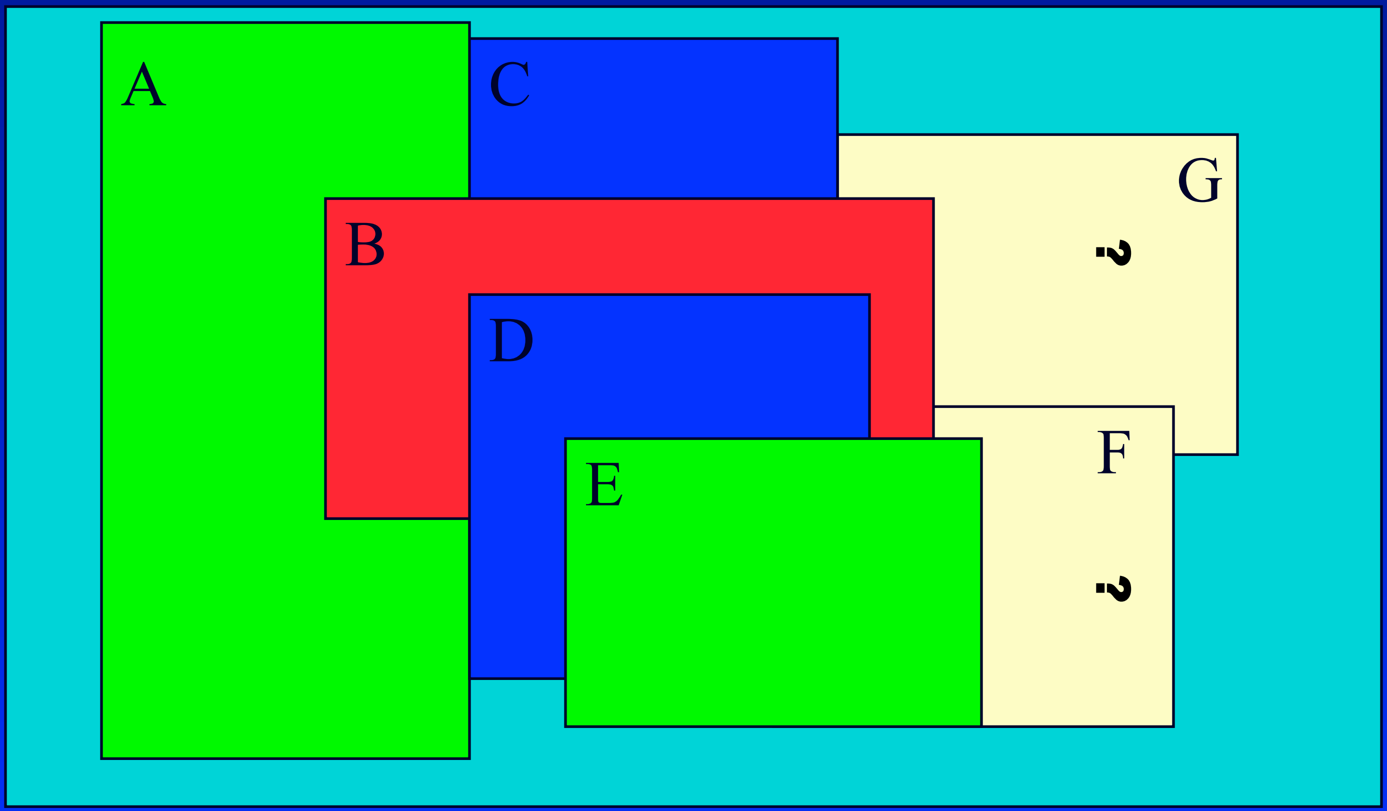

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)

# Analyzing Problem Structures

- some problem properties can be derived from the structure of the respective constraint graph
  - isolated sub-problems
    - no connections to other parts of the graph
    - can be solved independently
    - e.b. “islands” in map-coloring problems
    - dividing a problem into independent sub-problems reduces complexity tremendously
      - ideally from exponential to polynomial or even linear
  - tree
    - if the constraint graph is a tree, the CSP can be solved in time linear in the number of variables
    - sometimes solutions can be found by reducing a general graph to a tree
      - nodes are removed or collapsed

# CSP Example: Map Coloring (cont.)

- most-constrained-variable heuristic



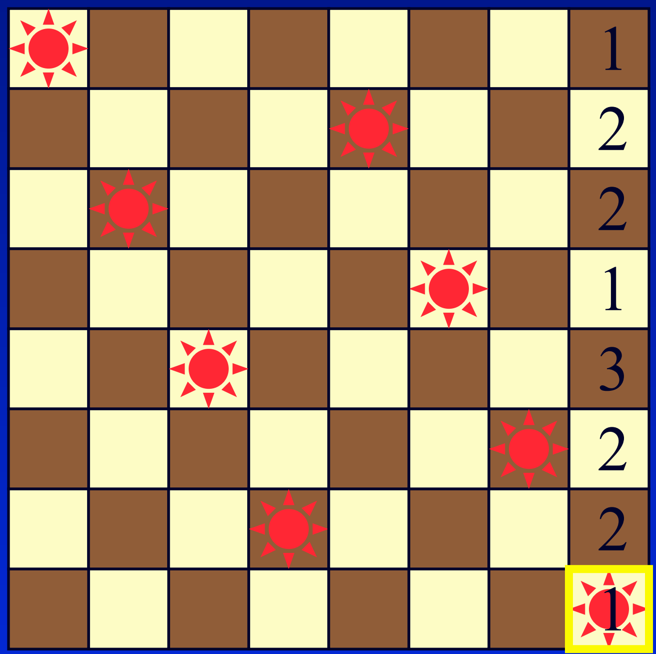

 

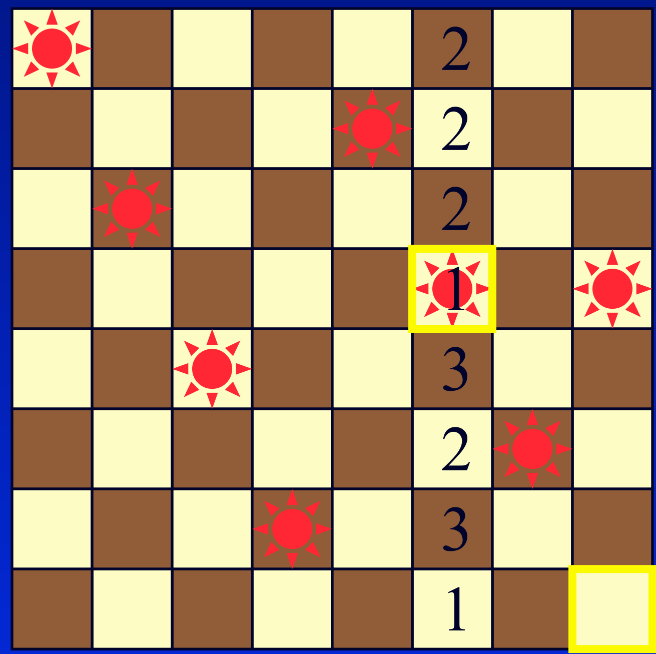
Wednesday, October 17, 12

# 8-Queens with Min-Conflicts

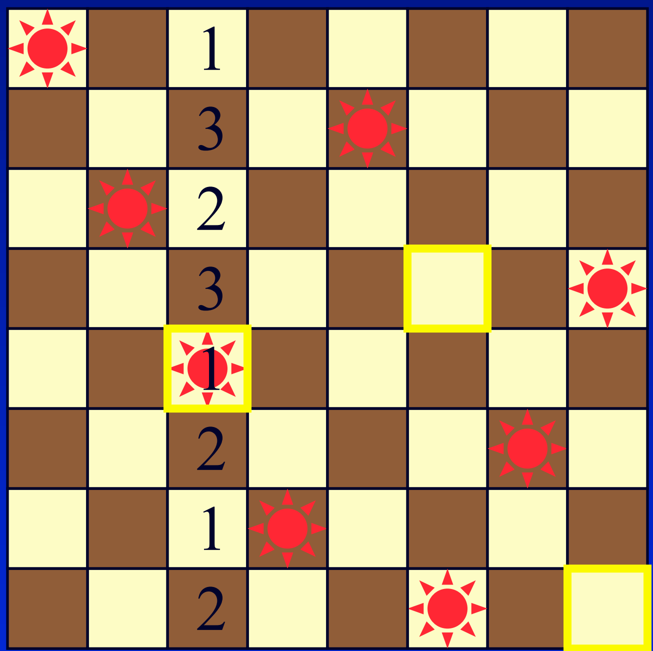
- one queen in each column
  - usually several conflicts
- calculate the number of conflicts for each possible position of a selected queen
- move the queen to the position with the least conflicts

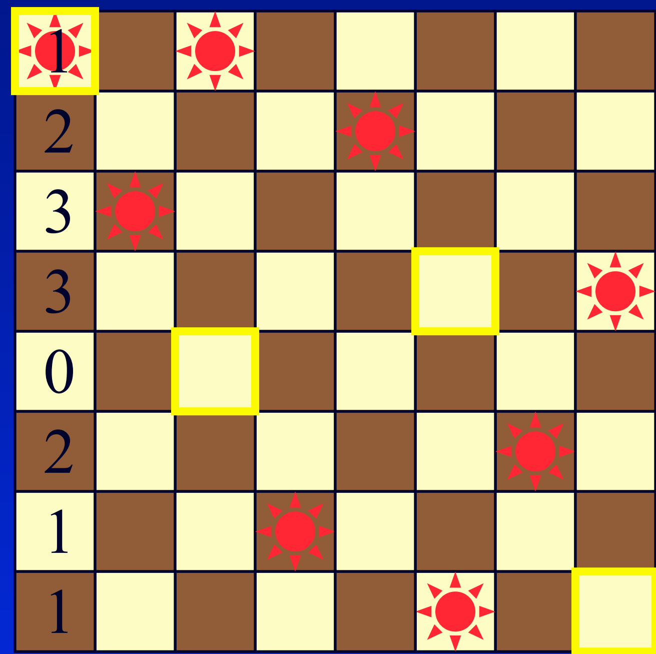
# 8-Queens Example 1



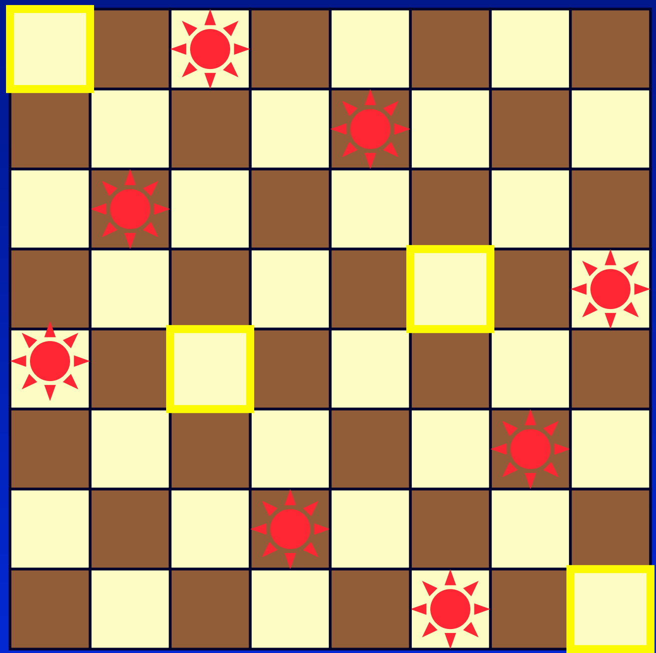



# 8-Queens Example 1



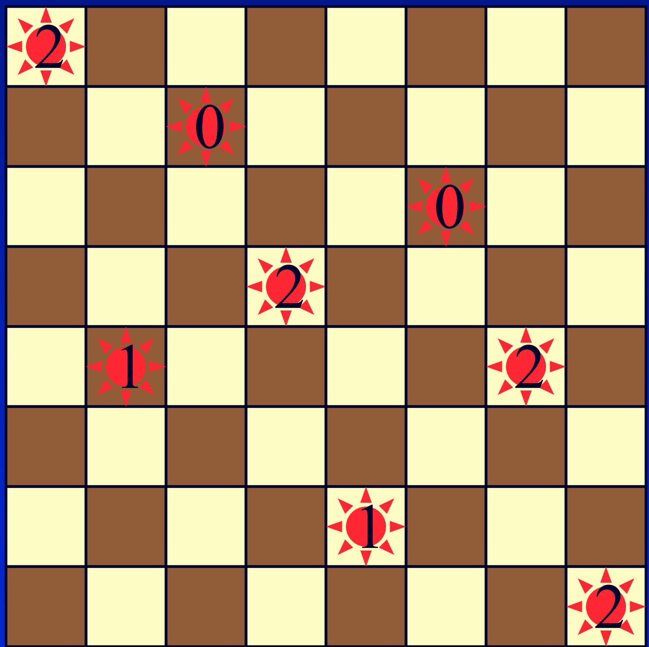
# 8-Queens Example 1

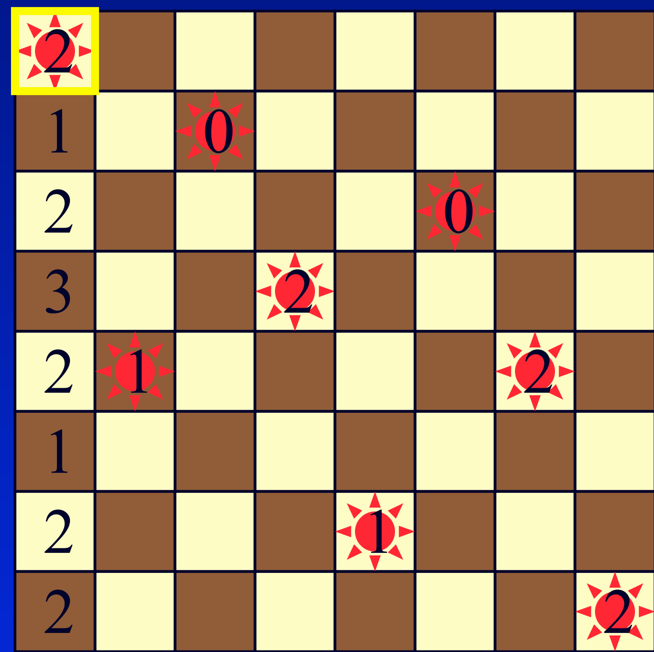


- solution found in 4 steps
- min-conflicts heuristic
- uses additional heuristics to select the “best” queen to move
  - try to move out of the corners
    - similar to least-constraining value heuristics

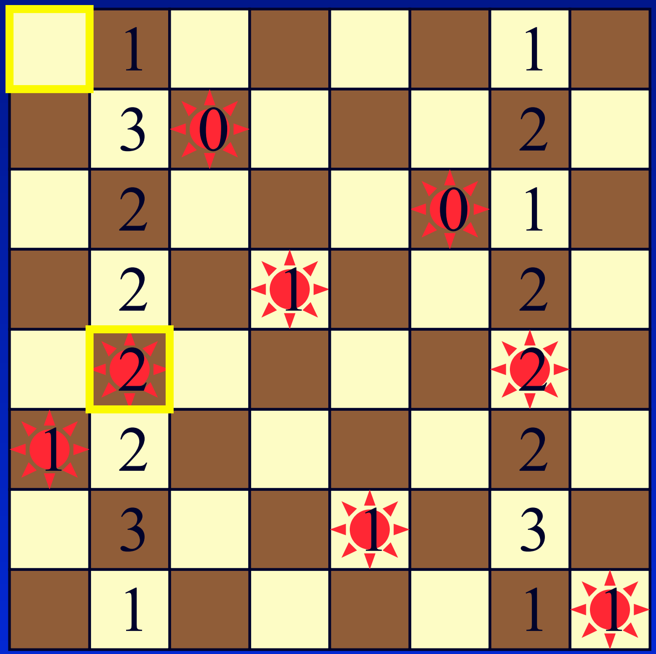
 

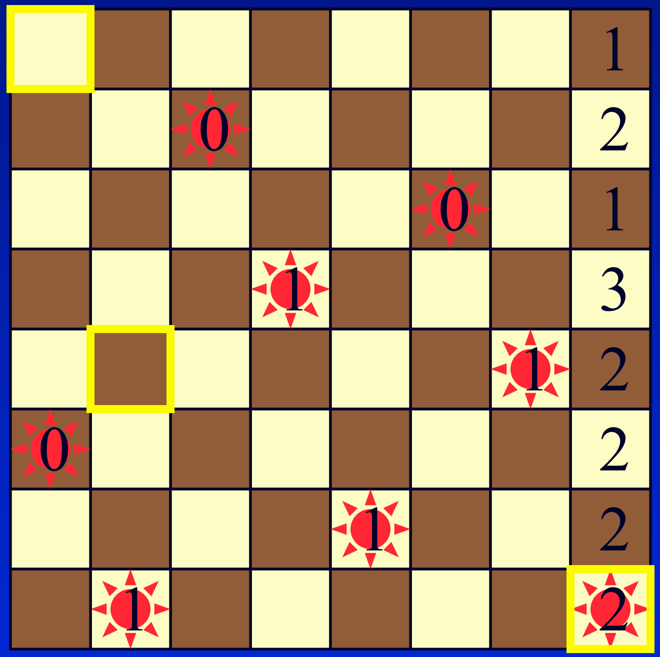
# 8-Queens Example 2



# 8-Queens Example 2



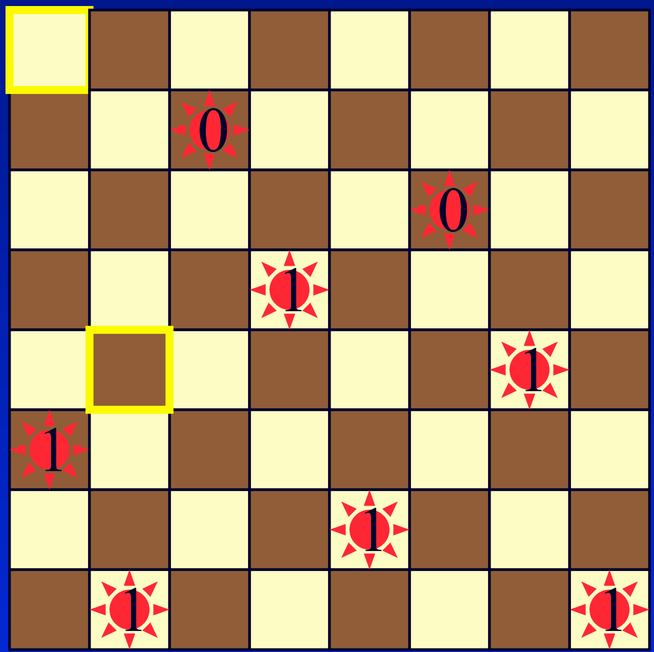



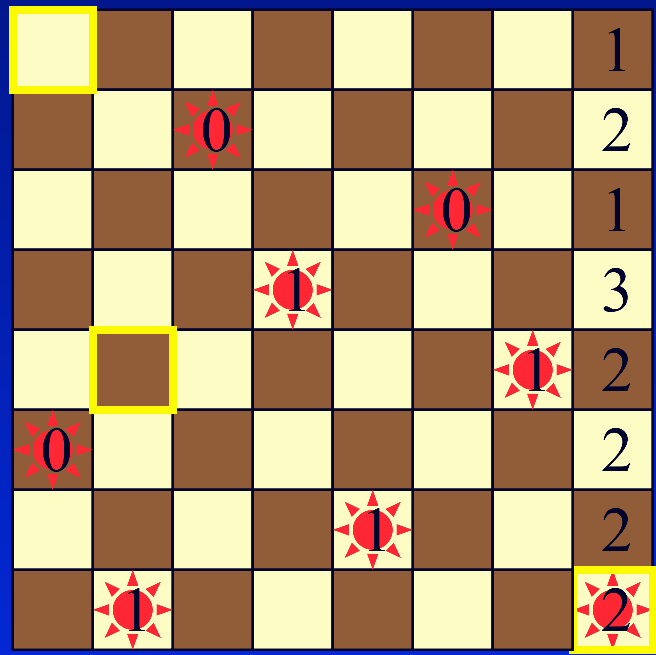





Wednesday, October 17, 12

# 8-Queens Example 2



# CSP Properties

- discrete variables over finite domains
  - relatively simple
    - e.g. map coloring, 8-queens
  - number of variable assignments can be dn
    - domain size d, n variables
  - exponential time complexity (worst-case)
  - in practice, generic CSP algorithms can solve problems much larger than regular search algorithms
- more complex problems may require the use of a constraint language
  - it is not possible to enumerate all combinations of values
  - e.g. job scheduling (“precedes”, “duration”)
- related problems are studied in the field of Operations Research
  - often continuous domains; e.g. linear programming

# Search in Continuous Spaces

- almost all algorithms discussed earlier are suited only for discrete spaces
  - some variations of hill climbing and simulated annealing work for continuous environments as well
- continuous state and action spaces have infinite branching factors
  - *really* bad for most conventional search algorithms
- many techniques for search in continuous spaces have been developed in other fields
  - the development of calculus by Newton and Leibniz provided the main tools

# Gradient Descent Search

- from the current state, select the direction with the steepest slope
  - the gradient of the objective function gives the magnitude and direction of the steepest slope
  - finding the maximum corresponds to solving an equation
    - in the landscape analogy, the mathematical maximum is often called a minimum since it is the lowest point
  - in many cases, it is not practical to calculate the *global* maximum
    - but it is often easy to compute the *local* maximum
- can be viewed as the inverse of hill climbing

# Search with Non-Deterministic Actions

- most of the earlier search algorithms assume an environment that is fully observable and deterministic
  - this allows off-line search
    - the agent can first do the calculations for the search until it finds the goal, and then pursue a particular path by executing the respective actions
- in non-deterministic environments, the agent needs to deal with contingencies
  - situations where important information is only available at the time the agent executes its actions
  - the solution to a problem then is not a sequence of actions, but a *contingency plan (strategy)*
    - it can contain nested if-then-else statements

# AND-OR Search Trees

- trees used to find contingent solutions for non-deterministic problems
- OR nodes express choices the agent has in each state
  - these correspond to the nodes in a deterministic search tree
- AND nodes reflect choices by the environment (contingencies)
  - the agent needs to prepare a plan for all potential choices, which corresponds to a set of AND-connected nodes

# Partially Observable Environments

- searching with no observation
  - sensor-less or conformant problems
- partial observations and percepts
  - a single percept may be associated with multiple states
    - the missing information may distinguish the states

# Belief States

- the belief-state space consists of all possible states that the agent knows of
  - in a fully observable environment, each belief state corresponds to exactly one physical state
- the belief state space contains every possible set of physical states
  - exponential with respect to the physical state size
  - many nodes in belief-state space may be unreachable
    - they do not correspond to a valid percept/state combination in the physical environment

# Search in Belief-State Space

- in belief-state space, search is fully observable
  - the agent knows its own belief state
  - there is no sensory input (in belief-state)
- the solution is always a sequence of actions for the belief-state
  - even if the actual environment is non-deterministic
  - the percept received after each action is predictable, since it is always empty
- the agent updates its belief state as information about the physical states becomes available
- practical approaches are known under various names
  - filtering, state estimation
  - many use probabilistic techniques

# Online Search and Unknown Environments

- in online search, the agent interleaves search and execution steps
  - often necessary in dynamic or non-deterministic environments
- in unknown environments, the agent has no choice but to perform online search
  - also known as *exploration problem*
- actions may be non-reversible
  - this leads to dead end, where no goal state is reachable
    - the agent is not necessarily “stuck”

# Online Local Search

- hill-climbing search is an online method
  - random restart may not work, however
    - the agent often can't just be moved to a random point in a real-world environment
- random walk
  - if the state space is finite, the agent will eventually find a goal or completely explore the state space
- combining hill-climbing with memory works better
  - store a current best estimate (heuristic) for each visited node
  - leads to an algorithm called *learning real-time A* (LRTA*)*
    - complete for finite, safely explorable environments

# Important Concepts and Terms

- agent
- A* search
- best-first search
- bi-directional search
- breadth-first search
- depth-first search
- depth-limited search
- completeness
- constraint satisfaction
- depth-limited search
- genetic algorithm
- general search algorithm
- goal
- goal test function
- greedy best-first search
- heuristics
- initial state
- iterative deepening search
- iterative improvement
- local search
- memory-bounded search
- operator
- optimality
- path
- path cost function
- problem
- recursive best-first search
- search
- space complexity
- state
- state space
- time complexity
- uniform-cost search