Announcements

- Coding Homework 1 will be released
  - Due 1/25 at 11:59pm
- Written Homework 1 will be released
  - Due 1/25 at 11:59pm

Optimization

Forest Agostinelli
University of South Carolina

Topics Covered in This Class

- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

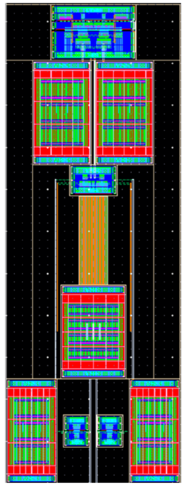
Outline

- Background
- State space search methods
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Evolutionary algorithms
- Constraint satisfaction
  - Backtracking
  - Forward checking

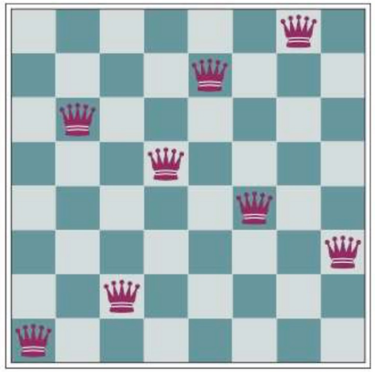
Optimization

- Find the best configuration (state) according to some objective function
- We want to either minimize cost of maximize value
- In many cases, we do not care about the path
  - The state itself is the solution!

Protein Structure Prediction
State: Bond angles
Objective Function: Free energy



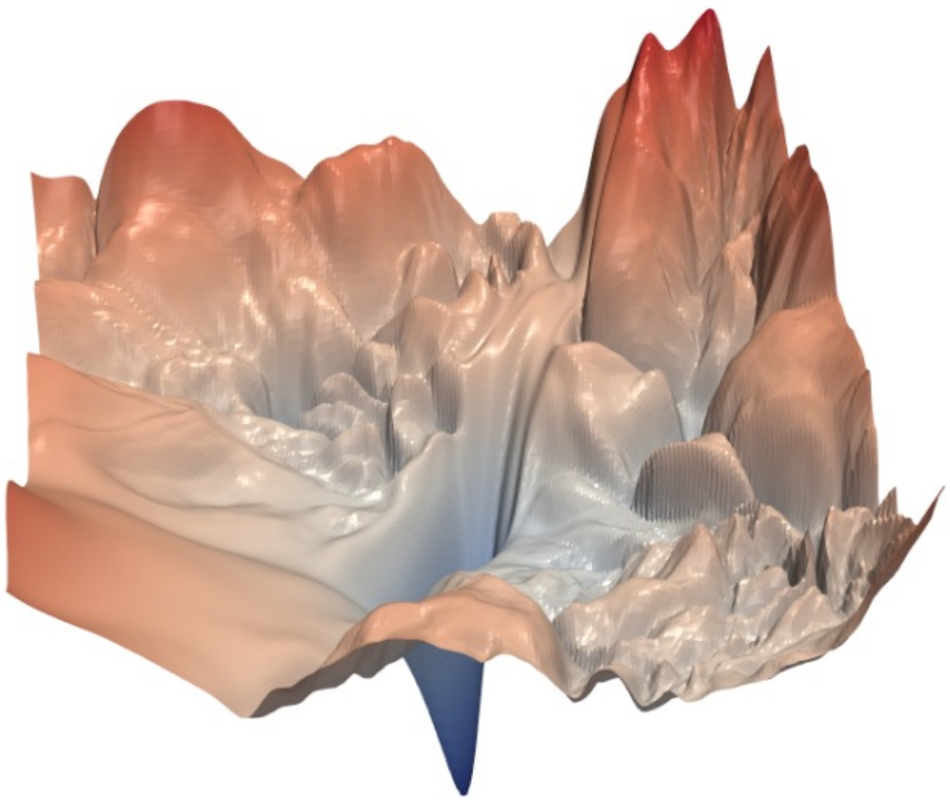
Circuit Design
State: Layout
Objective Function: Size, speed, power, etc.



N-Queens
State: Position on queens
Objective Function: How many queens are attacking each other

Optimization: A Visualization

- Objective function surface may be very jagged
- How to minimize cost when we only have local information available to us?



Optimization

- In this lecture we will focus on optimization of states that are represented by symbols instead of numbers
- However, there are many other optimization techniques that can be employed when the state space is represented by a mathematical expression
- We will talk about some of these techniques more when we talk about neural networks

Outline

- Background
- State space search methods
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Evolutionary algorithms
- Constraint satisfaction
  - Backtracking
  - Forward checking

Local Search

- Make local changes to a state or a small set of states in hopes of optimizing an objective function
- Do not remember paths
  - Very memory efficient
- In general, not guaranteed to find the best solution in a finite amount of time

Hill Climbing

- Look at every possible action you can take
- Transition to the state with the highest value

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

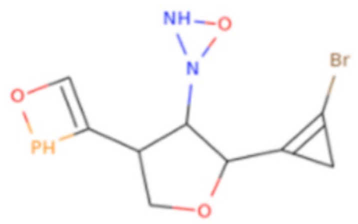
Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.

Hill Climbing: N-Queens

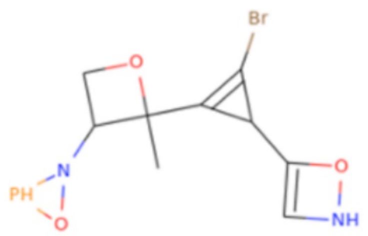
- Objective function: number of pairs of queens attacking each other
- Number is currently 17

Hill Climbing: Molecular Optimization

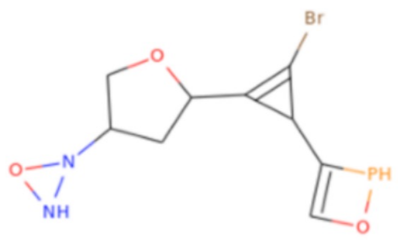
- Find a molecule that has a high quantitative estimate of drug-likeness (QED)
- States
  - Represent as a graph where vertices are atoms and edges are bonds
- Actions
  - Add/remove atom
  - Add/remove covalent bond
  - Atom donates an electron (ionic bond)



QED : 0.948 (C9H10BrN2O3P)



QED : 0.948 (C9H10BrN2O3P)

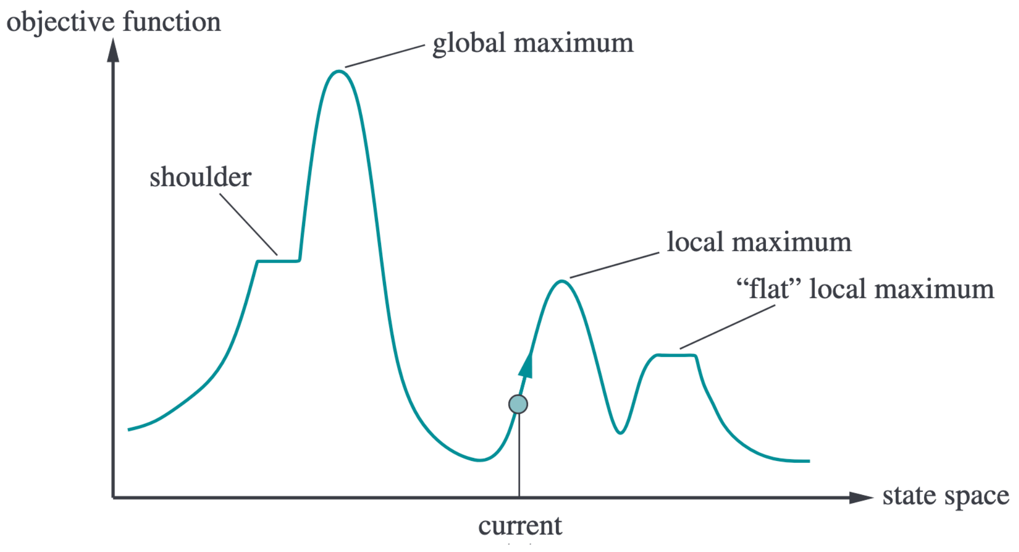


QED : 0.948 (C9H10BrN2O3P)

Leguy, Jules, et al. "EvoMol: a flexible and interpretable evolutionary algorithm for unbiased de novo molecular generation." Journal of cheminformatics 12.1 (2020): 1-19.

Hill Climbing: Quick Quiz

1. You are doing hill climbing search. In which state does the search terminate?
2. Hill climbing looks ahead beyond its immediate neighbors (T/F)
3. Hill climbing search is always guaranteed to find a globally optimal solution in finite time (T/F)



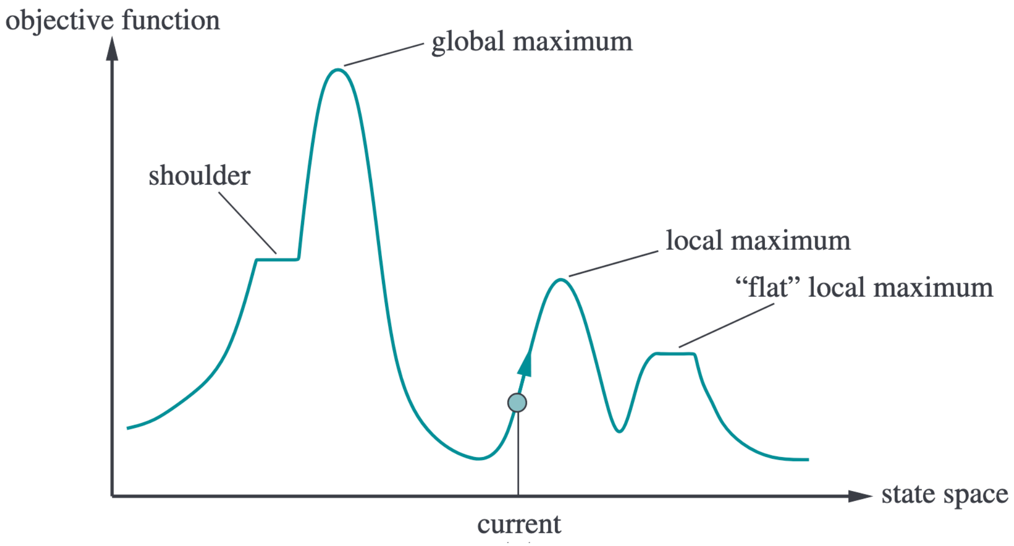
Maximize value

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.

Hill Climbing: 1D State Space Example

- **Global maximum**
  - The highest peak
- **Local maximum**
  - Higher than its neighboring states but lower than the global maximum
- **Plateau**
  - A flat area
  - Can be a shoulder or a flat local maximum



Maximize value

Hill Climbing with Random Restarts

- Hill climbing is successful for the 8-queens problem only 14% of the time (gets stuck 86% of the time)
- If hill-climbing gets stuck at a plateau or a local maximum, or if a time limit is reached, restart from a randomly generated state
- 8-queens needs roughly 6 restarts to solve the problem
- Can be very effective in practice
  - Even for 3 million queens, it can find a solution in seconds
- Given infinite time, it will find a solution with probability 1
  - It will eventually randomly generate the goal state as the initial state

Tabu Search

- Add k recently visited states to a tabu list
- Excludes these states from being visited again
- Can help escape from plateaus can local maxima

Simulated Annealing

- Accept transitioning to states with a lower value with some probability
- The probability is initially high, but becomes lower over time
- Can be used for VLSI layouts, airline scheduling, etc.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

Figure 4.4 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. The *schedule* input determines the value of the "temperature" T as a function of time.

<table>
<tr><td colspan="2" rowspan="2">$e^{\Delta E / T}$</td><td colspan="2">Temperature T</td></tr>
<tr><td>High</td><td>Low</td></tr>
<tr><td rowspan="2">|ΔE|</td><td>High</td><td>Medium</td><td>Low</td></tr>
<tr><td>Low</td><td>High</td><td>Medium</td></tr>
</table>

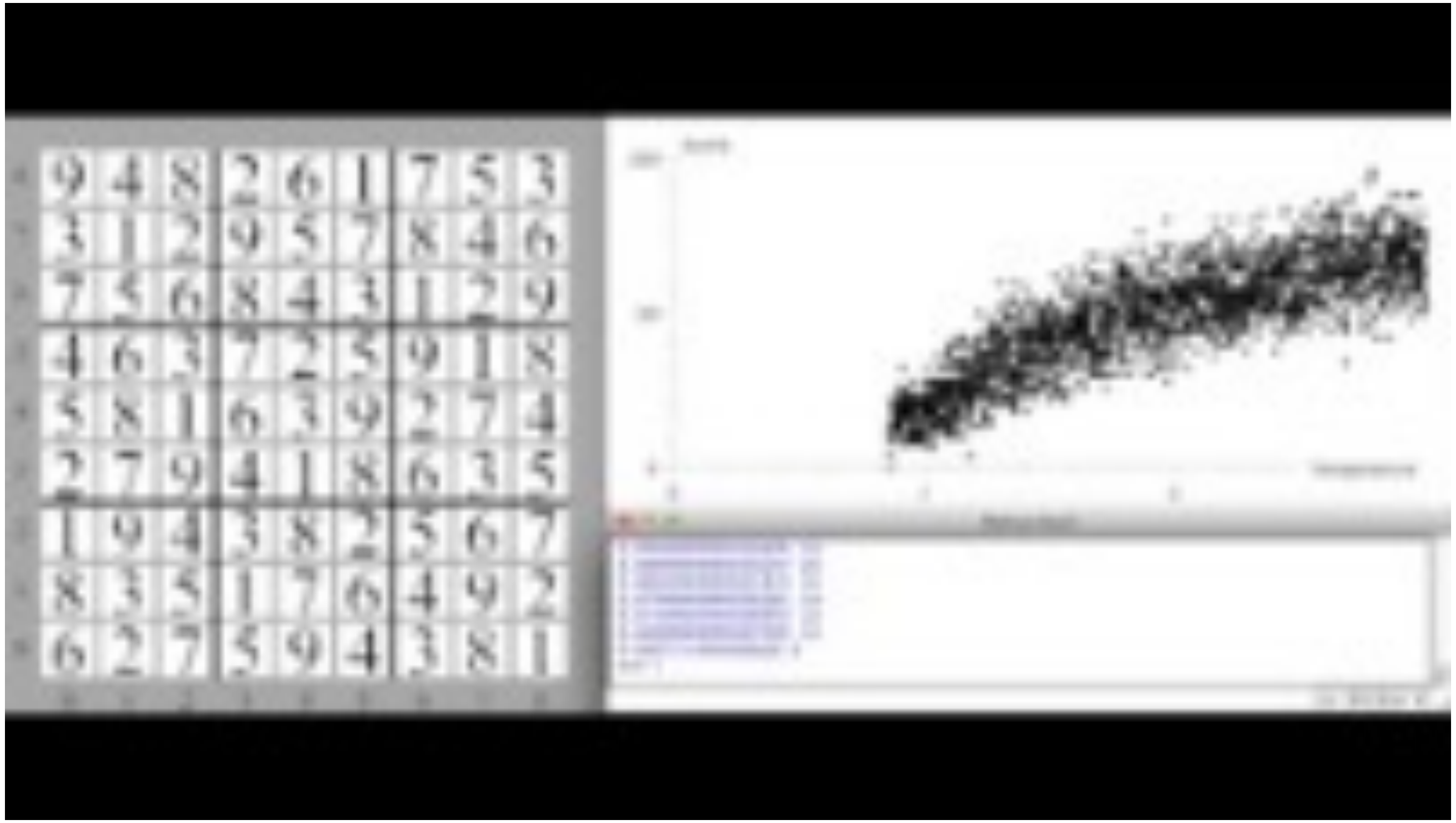
Probability of accepting next state

Simulated Annealing

- If you decrease T slowly enough, will find global maximum with probability 1
  - Can take a very long time!
- Can work well in practice
  - Was used to solve VLSI layout problems in the 1980s

Solving Sudoku with Simulated Annealing

- Takes 3 hours. How can we do this faster?

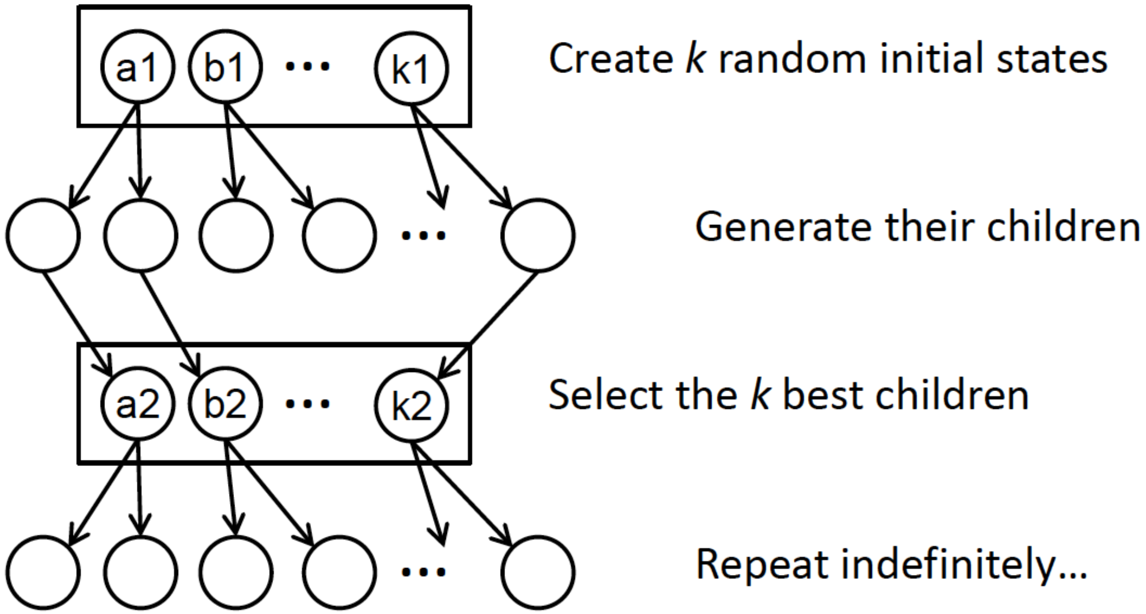


Local Beam Search

- Instead of keeping track of just one state, we keep track of k
- At each iteration, we generate all the successors of all the k states
- If any one is a goal, then return solution
- Else, select the k best and repeat

Local beam search

- Possibly better than running k independent searches
  - Concentrates on promising states
- May result in all states being concentrated in the same place, resulting in redundancies
- Ways to improve?



Stochastic Local Beam Search

- Instead of choosing the top k, choose k states with probability proportional to their value
- Can alleviate problem of all k states collapsing to the same state

Evolutionary Algorithms

- Maintain a “population” (collection of states) that produces “offspring” (new states)
- The “fittest” states (those with the highest value) are more likely to continue to the next generation
- Stochastic beam search can be posed as an instance of evolutionary algorithms
  - Only one parent

Evolutionary Algorithms

- The manner in which you represent the state can greatly affect the outcome

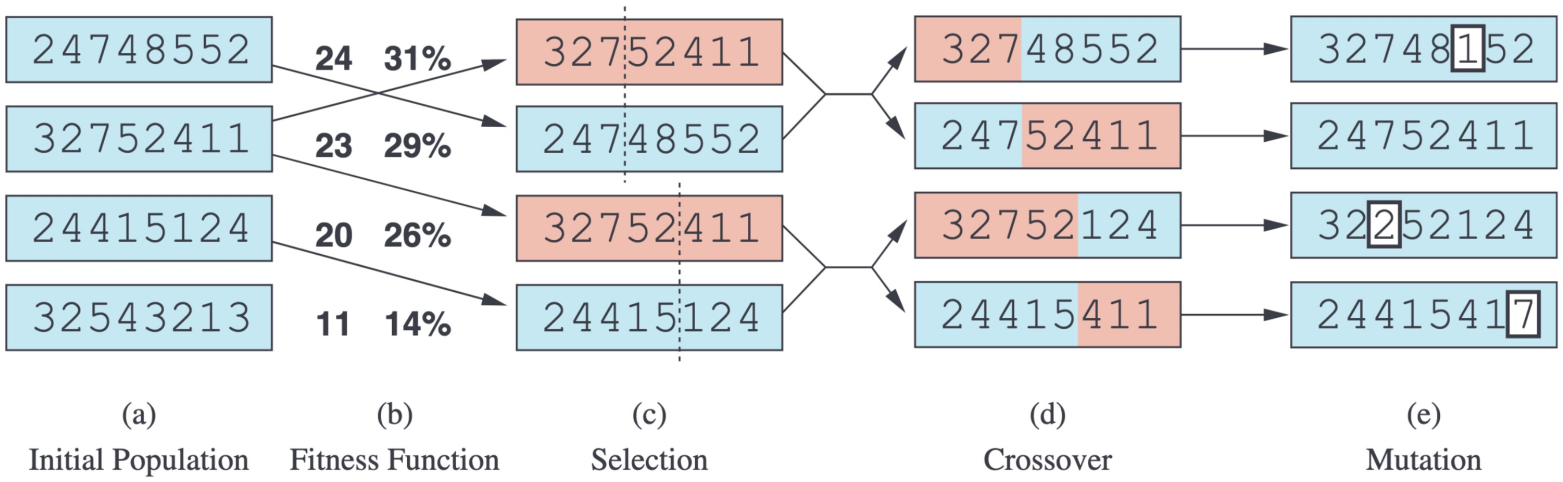
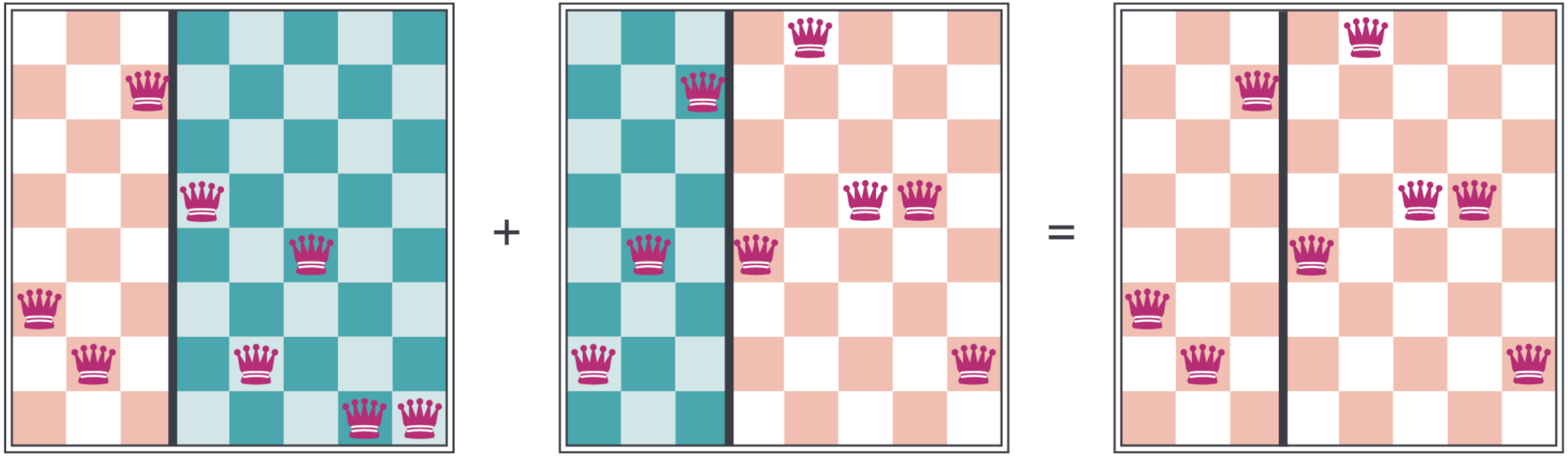


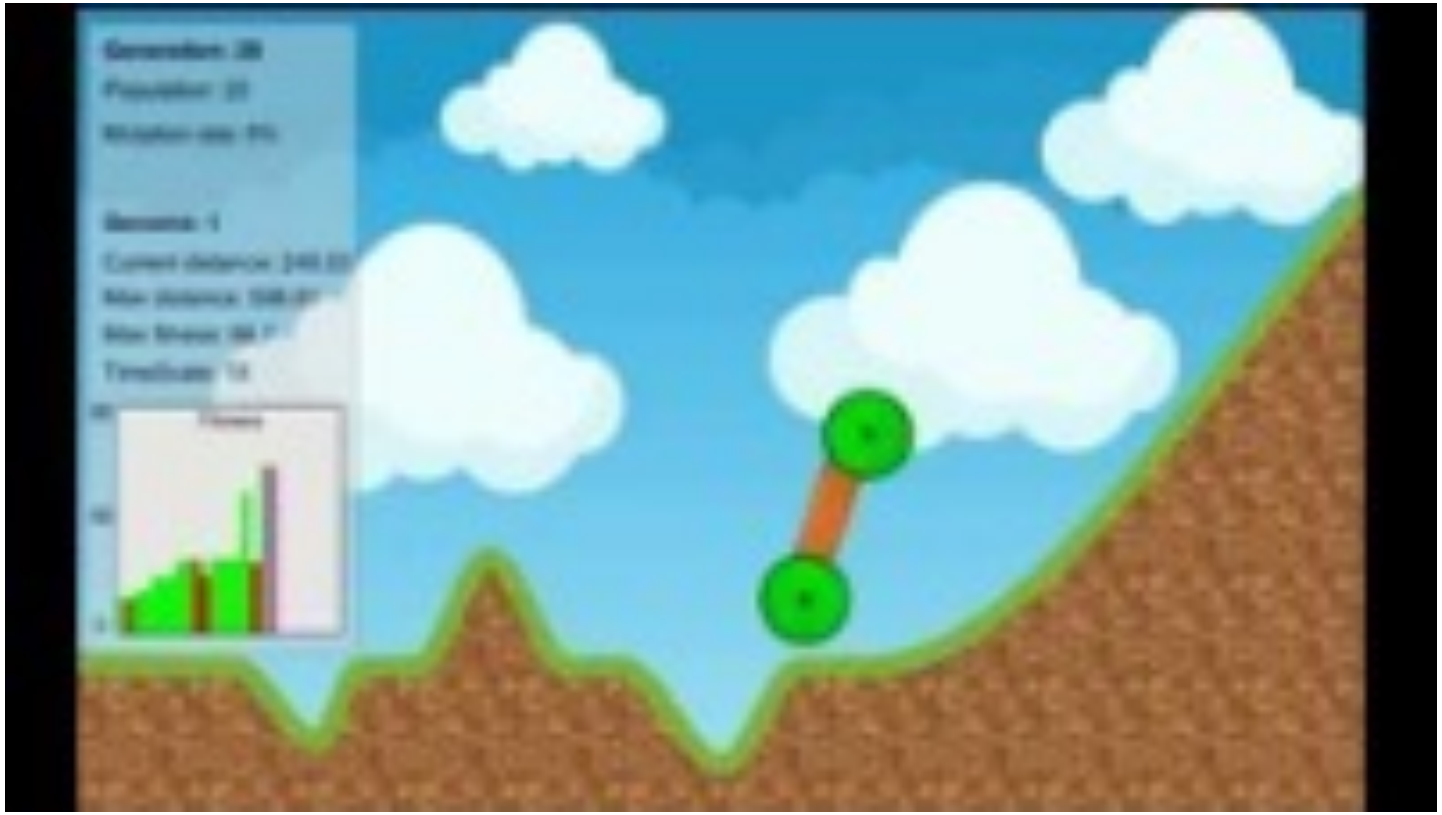
Figure 4.5 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Evolutionary Algorithms: N-Queens



Evolutionary Algorithms: Car Design

- Representation of the state?
- Crossover?

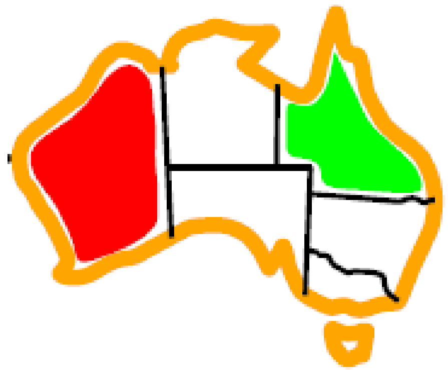


Outline

- Background
- State space search methods
  - Hill climbing
  - Simulated annealing
  - Local beam search
  - Evolutionary algorithms
- Constraint satisfaction
  - Backtracking
  - Forward checking

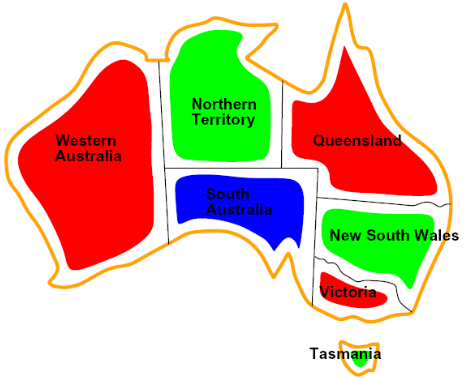
Exploiting State Structure

- So far, we have thought of states as “atomic” (indivisible)
- All we could do was map a state to a cost or value
- However, there may be valuable information in the structure of the problem
- For example, given an incomplete map for the map coloring problem, how can we prune the search space?



Constraint Satisfaction Problems (CSPs)

- On a high level, CSPs are problems that require assignments to variables where the assignments to those variables are subject to some constraints
  - Graph coloring
  - Scheduling
    - When to meet your friends
    - Classrooms
    - Job shops
    - Airplanes
  - Sudoku
  - N-queens



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

Constraint Satisfaction Problems

- $\mathcal{X}$ is a set of variables
- $\mathcal{D}$ is a set of domains, one for each variable
  - Allowable values
- $\mathcal{C}$ is a set of constraints that specify allowable combinations of variables
  - A tuple

CSPs: Assignments and Solutions

- Consistent assignment: An assignment to variables that does not violate constraints
- Complete assignment: Every variable is assignment a value
- Partial assignment: One that leaves some variables unassigned
- Partial solution: Partial assignment that is consistent
- Solution: A consistent and complete assignment

Map Coloring Example

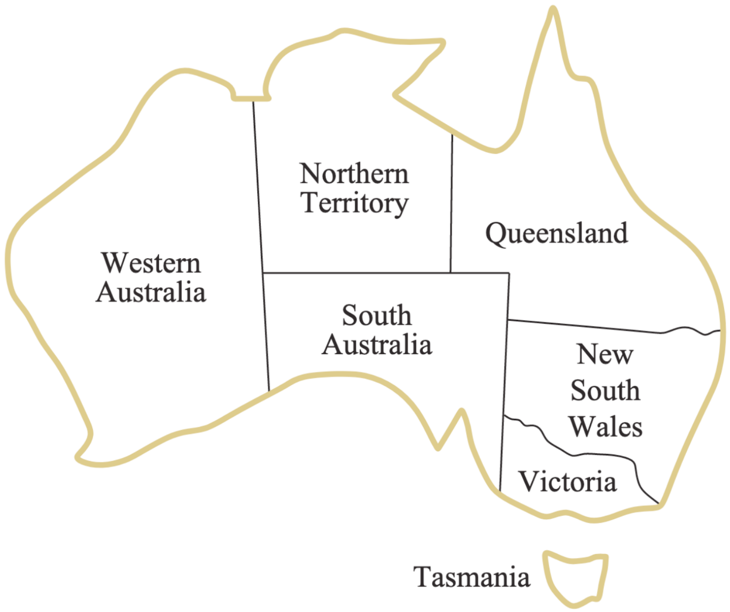
- Variables: $\mathsf{WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T}$
- Domains: $\mathsf{D} = \{\mathsf{red}, \mathsf{green}, \mathsf{blue}\}$
- Constraints: adjacent regions must have different colors

  Implicit: $\mathsf{WA} \neq \mathsf{NT}$

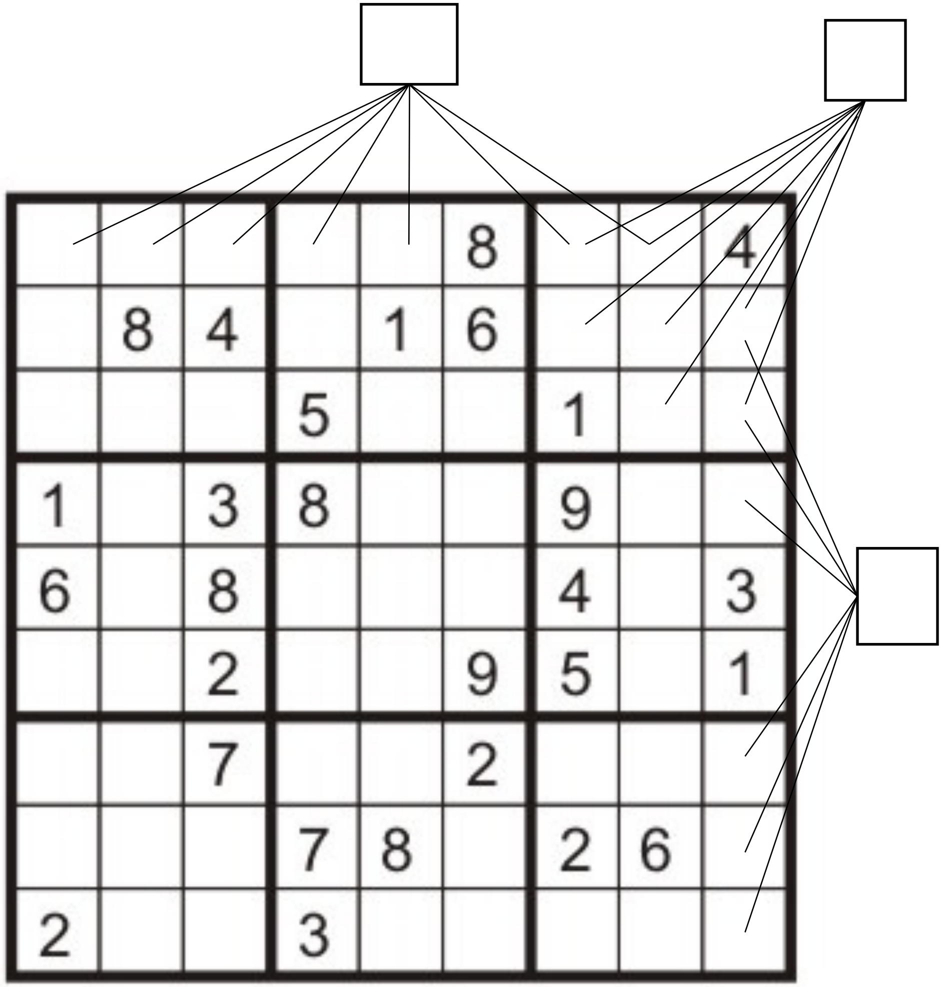
  Explicit: $(\mathsf{WA}, \mathsf{NT}) \in \{(\mathsf{red}, \mathsf{green}), (\mathsf{red}, \mathsf{blue}), \ldots\}$

- Solutions are assignments satisfying all constraints, e.g.:

$$\{\mathsf{WA{=}red,\ NT{=}green,\ Q{=}red,\ NSW{=}green,\ V{=}red,\ SA{=}blue,\ T{=}green}\}$$



Sudoku Example



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

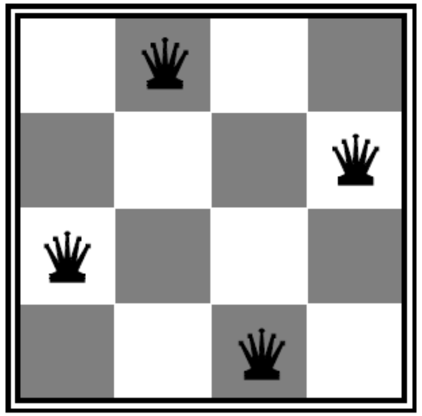
9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

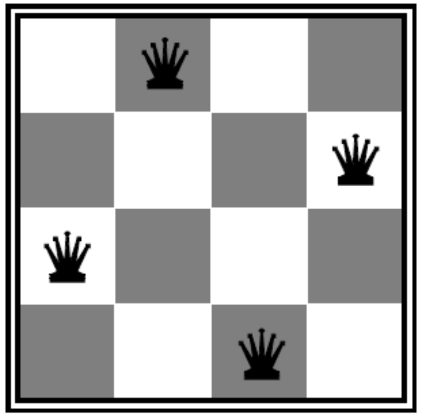
N-Queens Example

- Variables:
- Domains:
- Constraints



N-Queens Example

- Formulation 1:
  - Variables: Squares
  - Domains: 0/1 indication of queen
  - Constraints: Which combinations are allowed and there must be N total queens

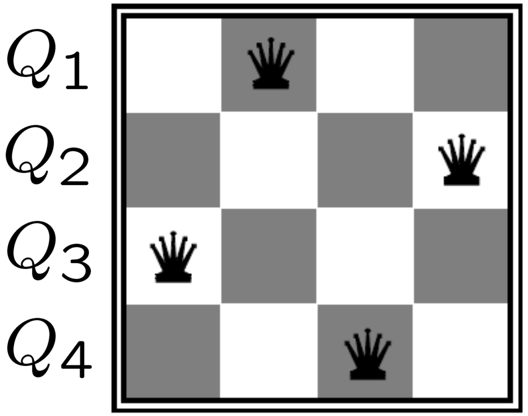


$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

N-Queens Example

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

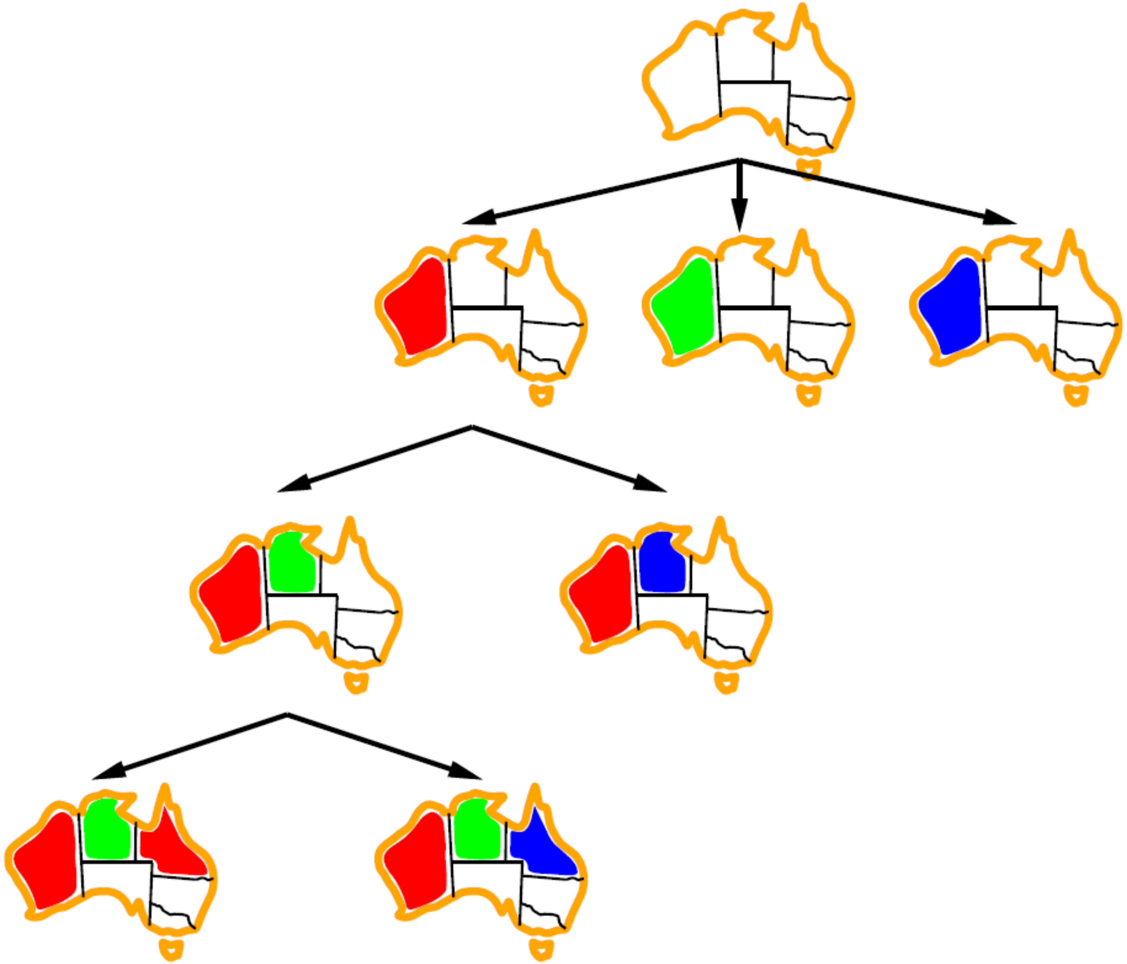


Implicit: $\forall i, j \ \text{non-threatening}(Q_i, Q_j)$

Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$

$\cdots$

Backtracking Search



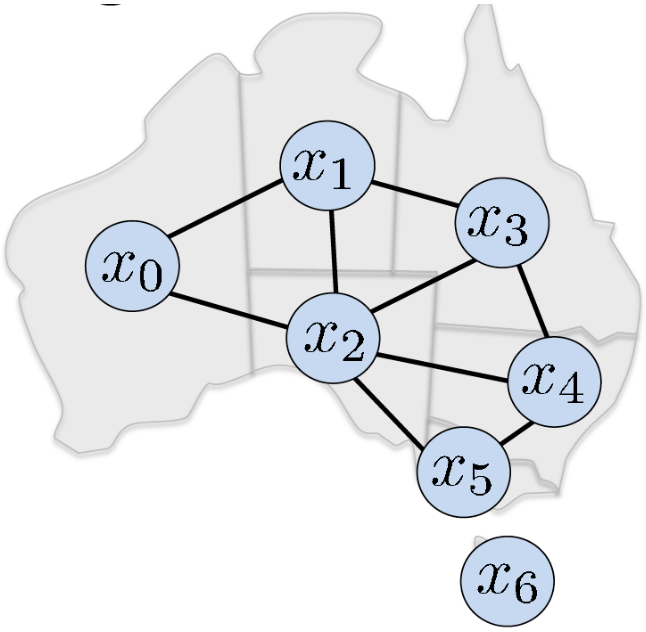
Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

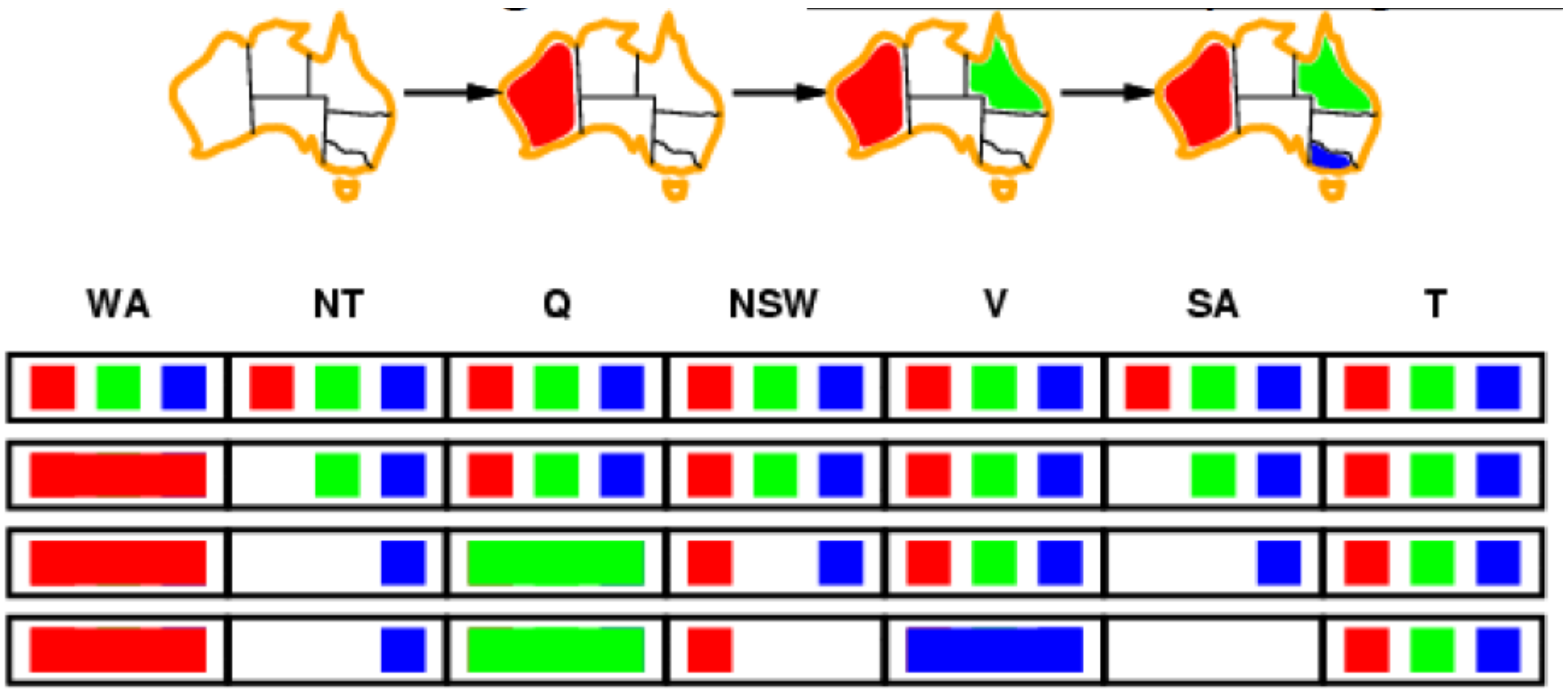
Inference in CSPs

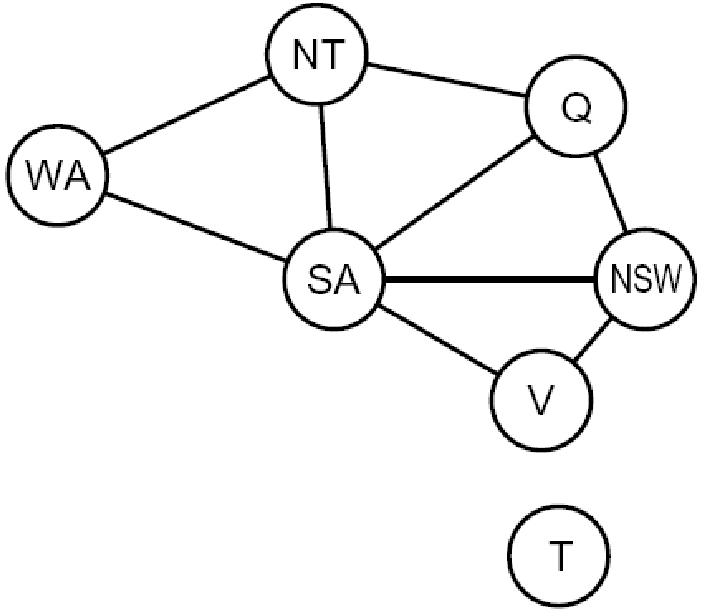
- Make use of a constraint graph
  - Nodes: variables
  - Edges: Connects variables that participate in a constraint
- Gives us an intuitive representation
- Makes it easy to prune large parts of the state space



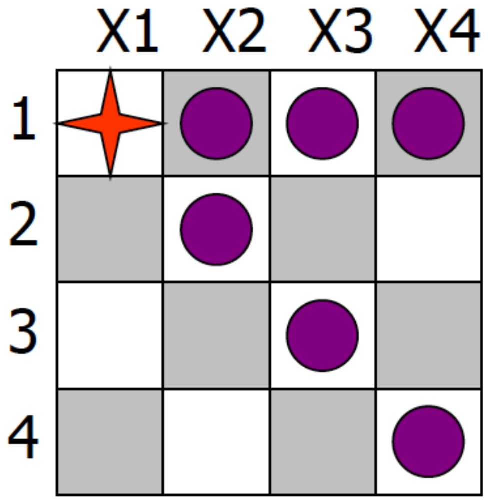
Inference in CSPs: Forward Checking

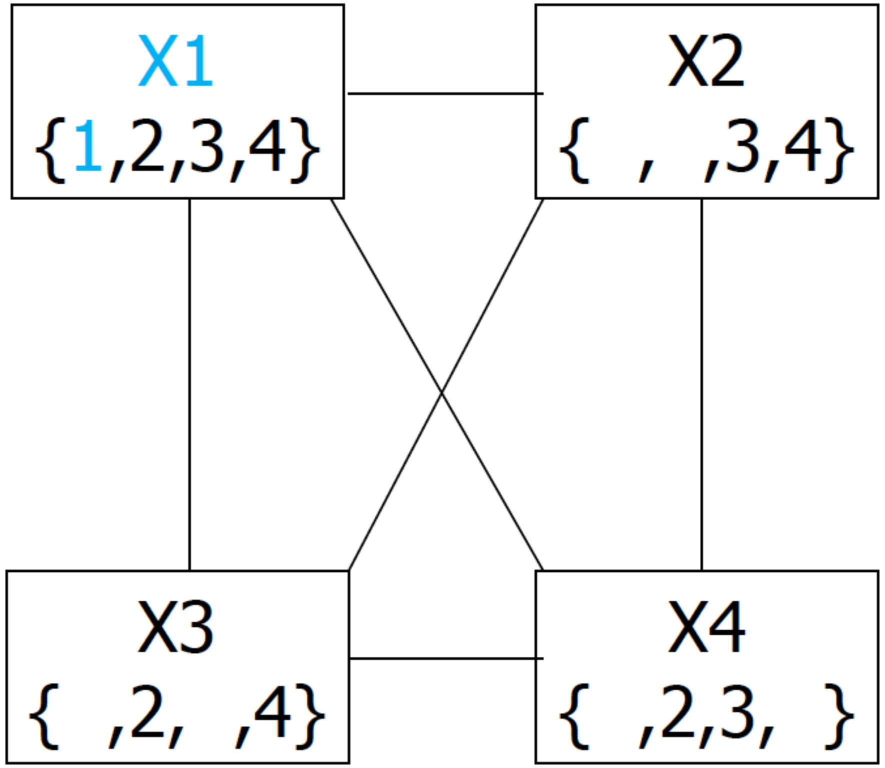
- Given an assignment to a variable, what can we infer?
- Forward checking: Remove values from domains that violate a constraint





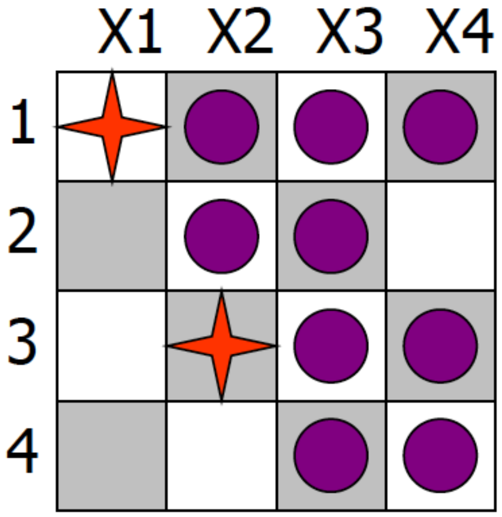
Forward Checking: N-Queens

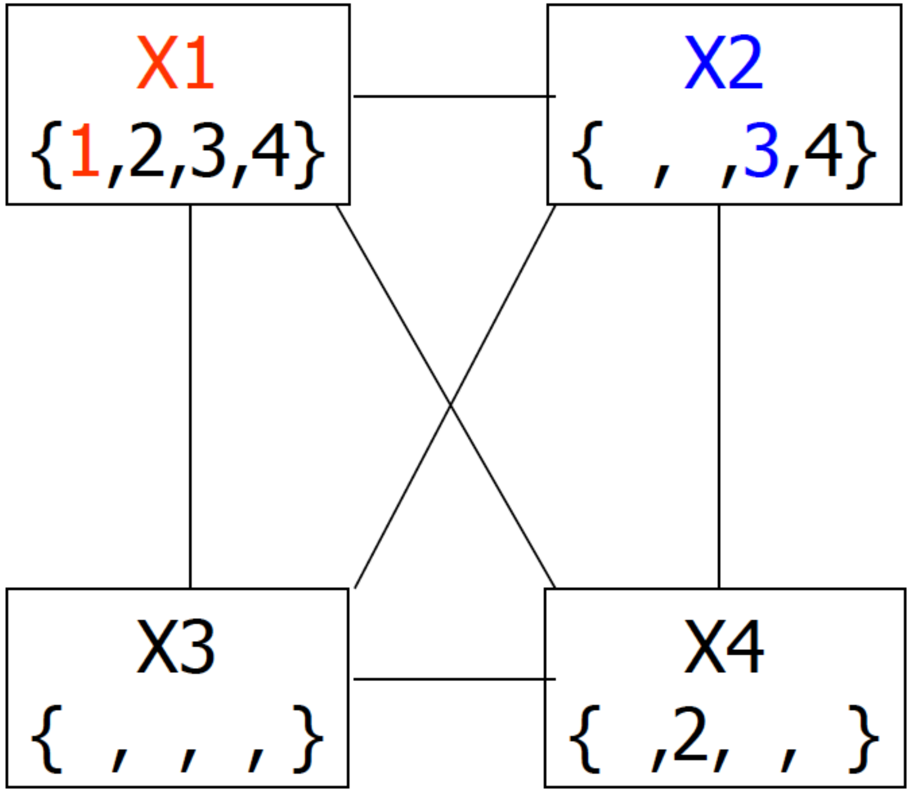




Red = value is assigned to variable
Blue = most recent variable/value pair

Forward Checking: N-Queens

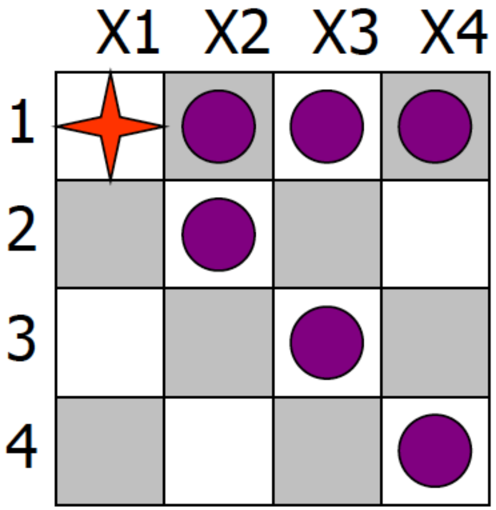


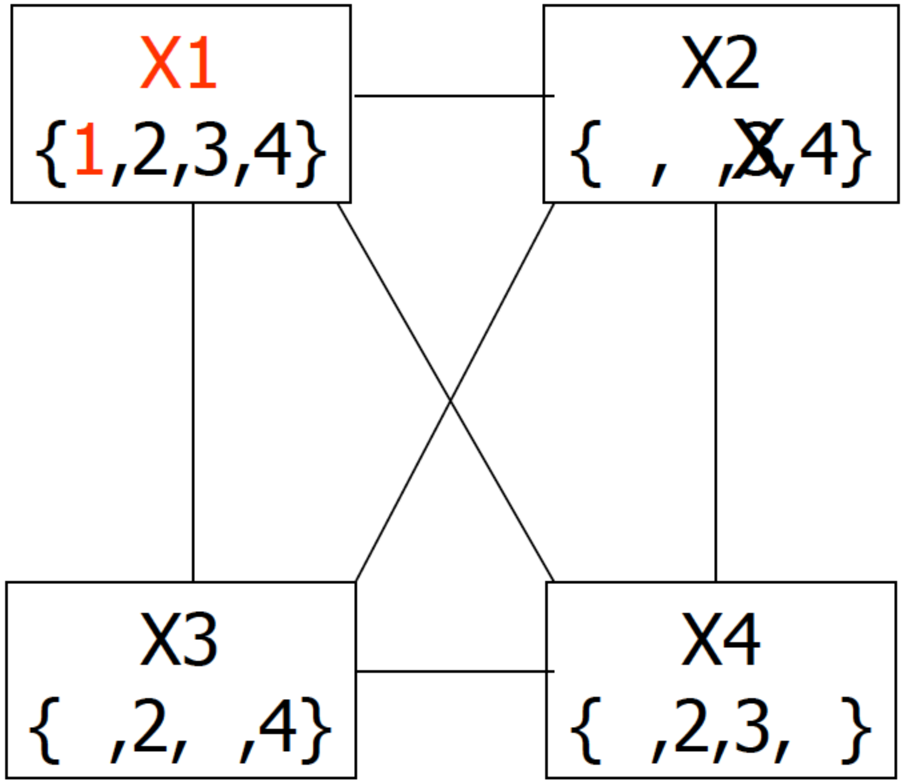


Red = value is assigned to variable

Blue = most recent variable/value pair

Forward Checking: N-Queens

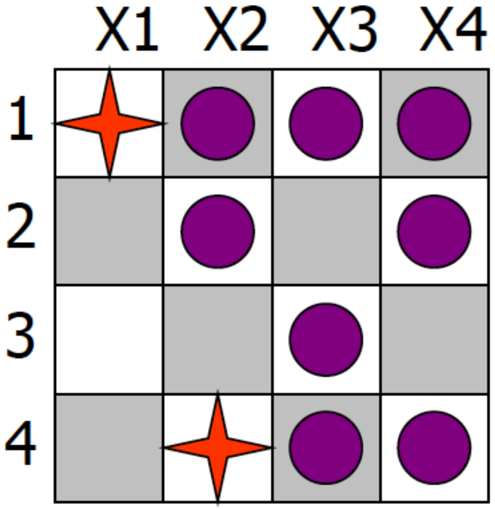


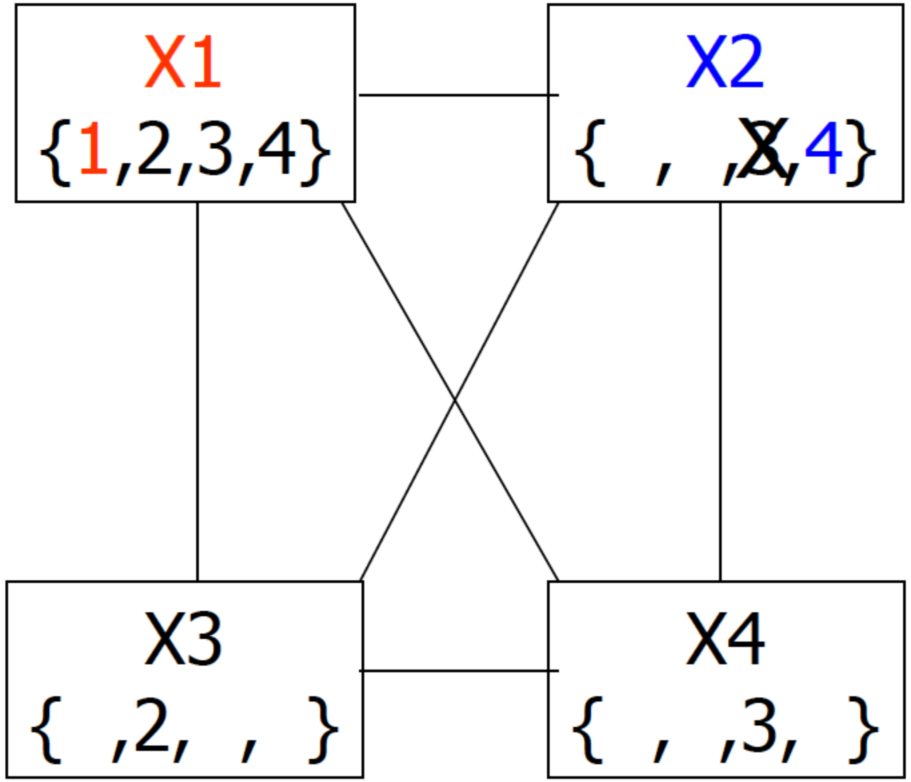


Red = value is assigned to variable

X = value led to failure

Forward Checking: N-Queens



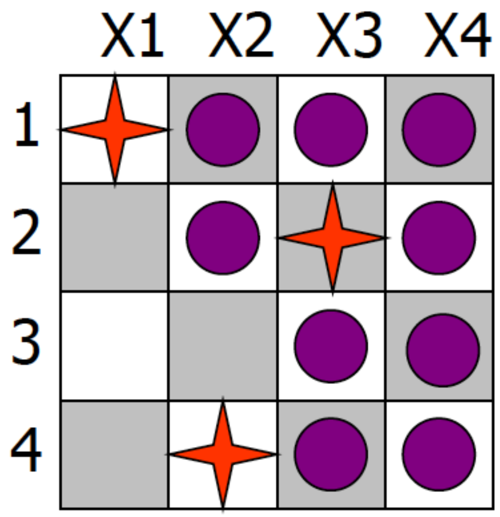


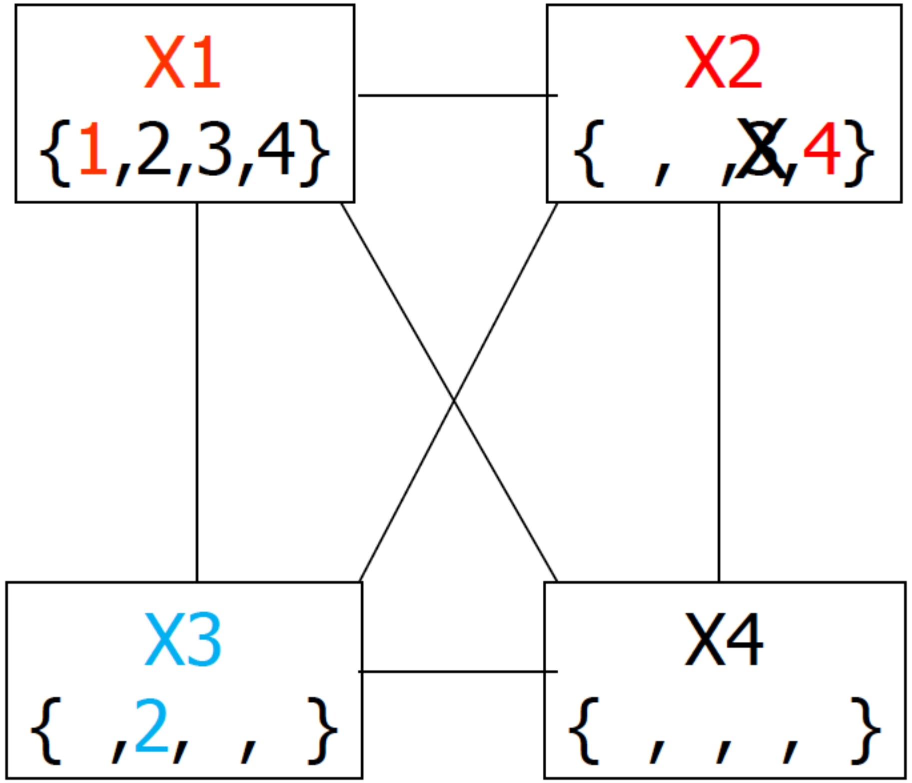
Red = value is assigned to variable

Blue = most recent variable/value pair

X = value led to failure

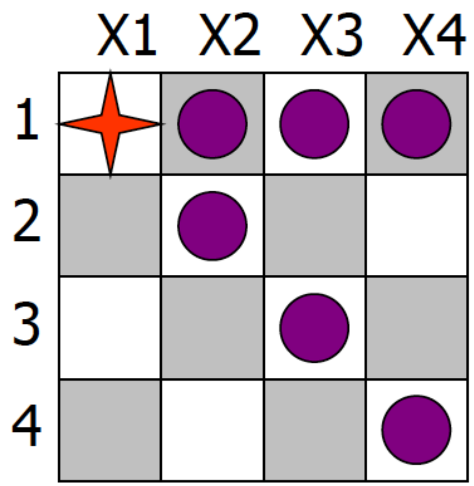
Forward Checking: N-Queens

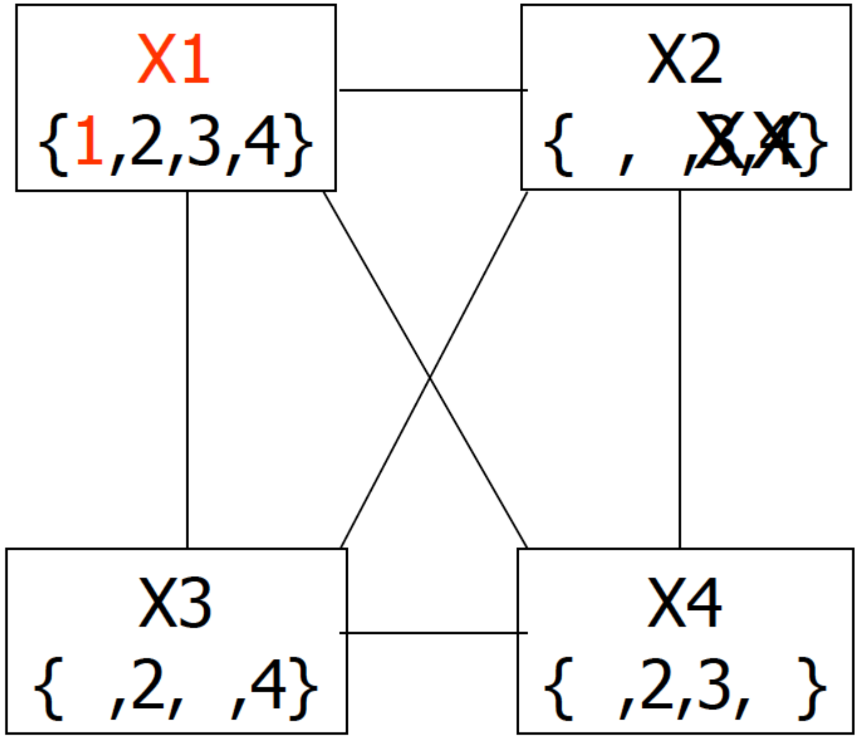




Red = value is assigned to variable
Blue = most recent variable/value pair
X = value led to failure

Forward Checking: N-Queens

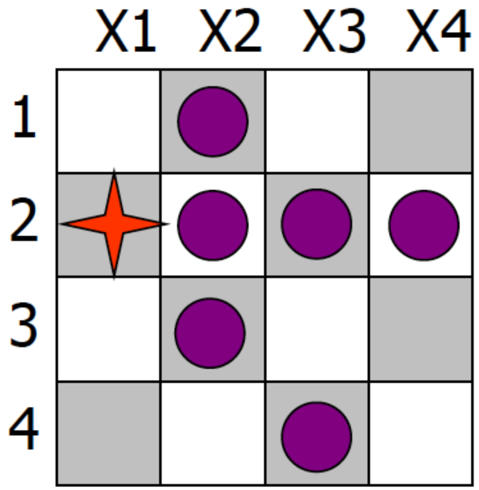


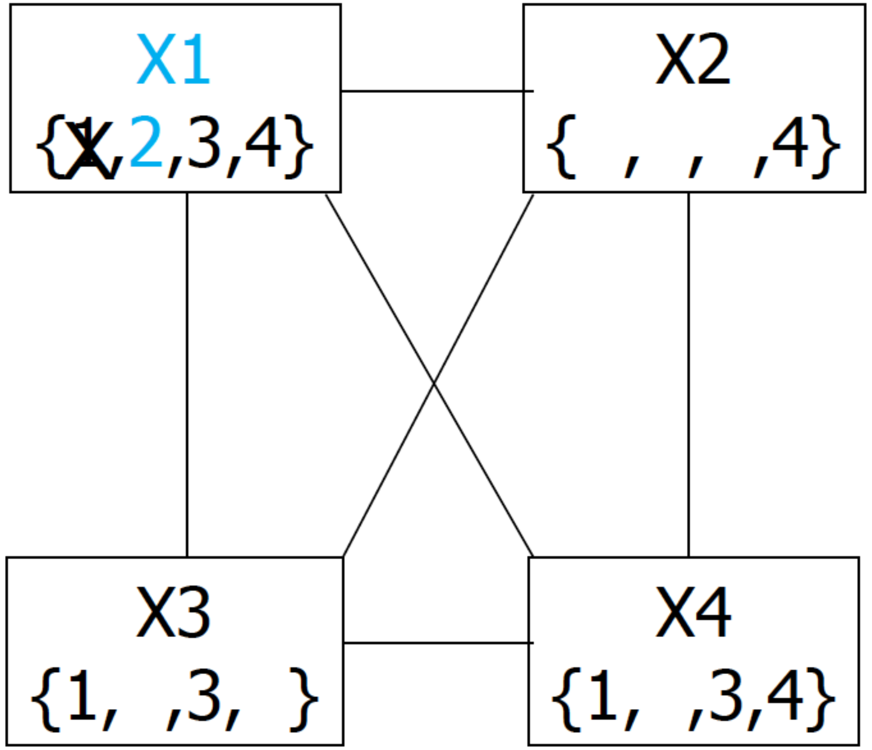


Red = value is assigned to variable

X = value led to failure

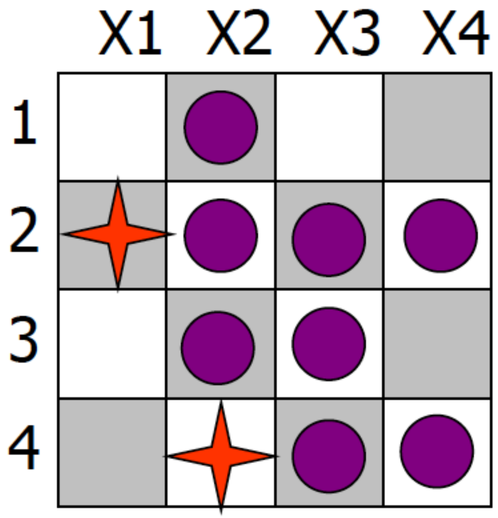
Forward Checking: N-Queens

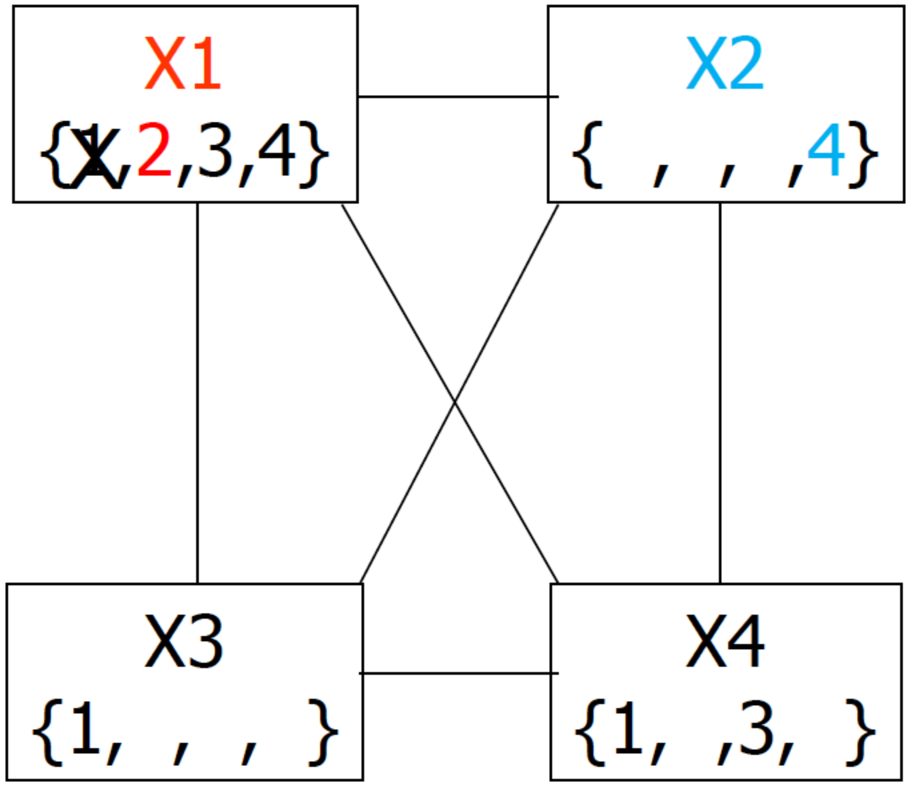




Red = value is assigned to variable
Blue = most recent variable/value pair
X = value led to failure

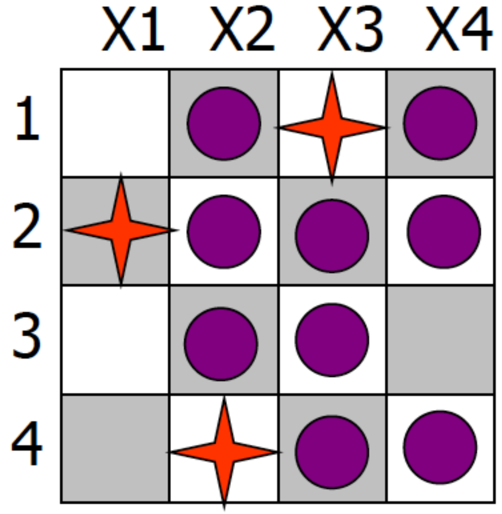
Forward Checking: N-Queens

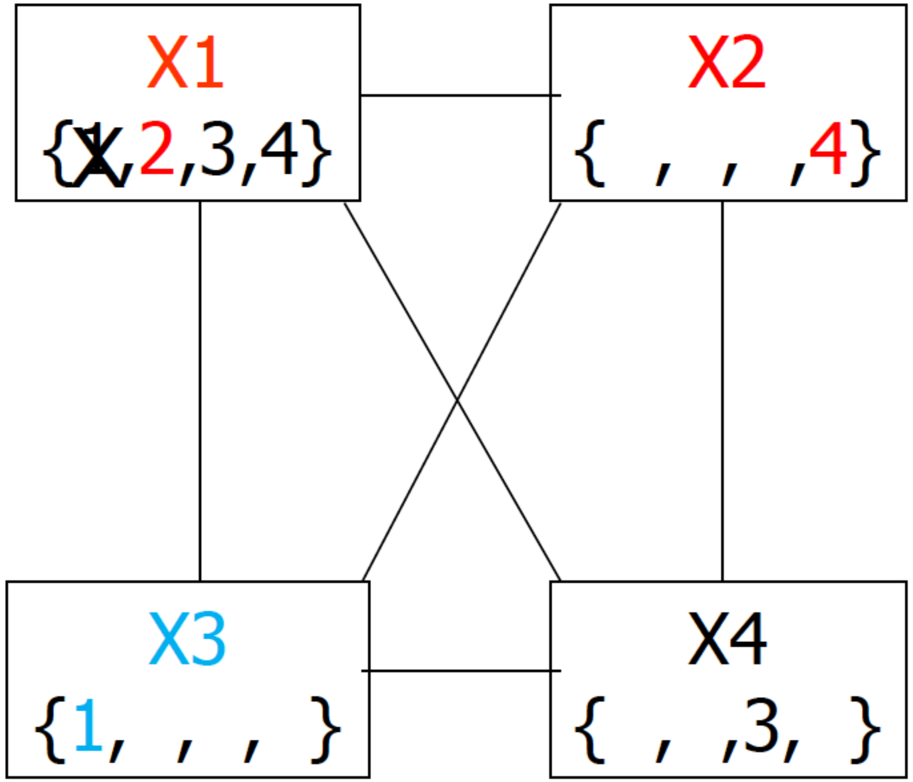




Red = value is assigned to variable
Blue = most recent variable/value pair
X = value led to failure

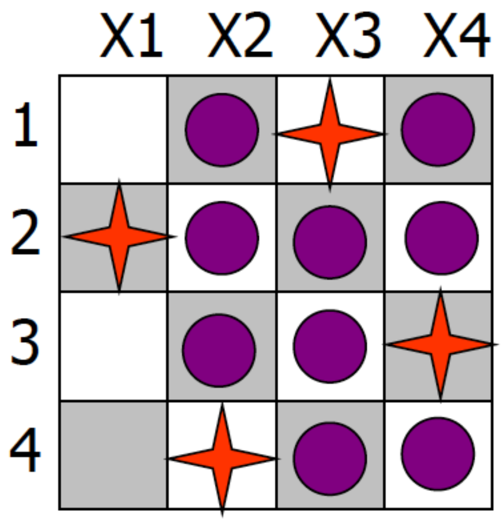
Forward Checking: N-Queens

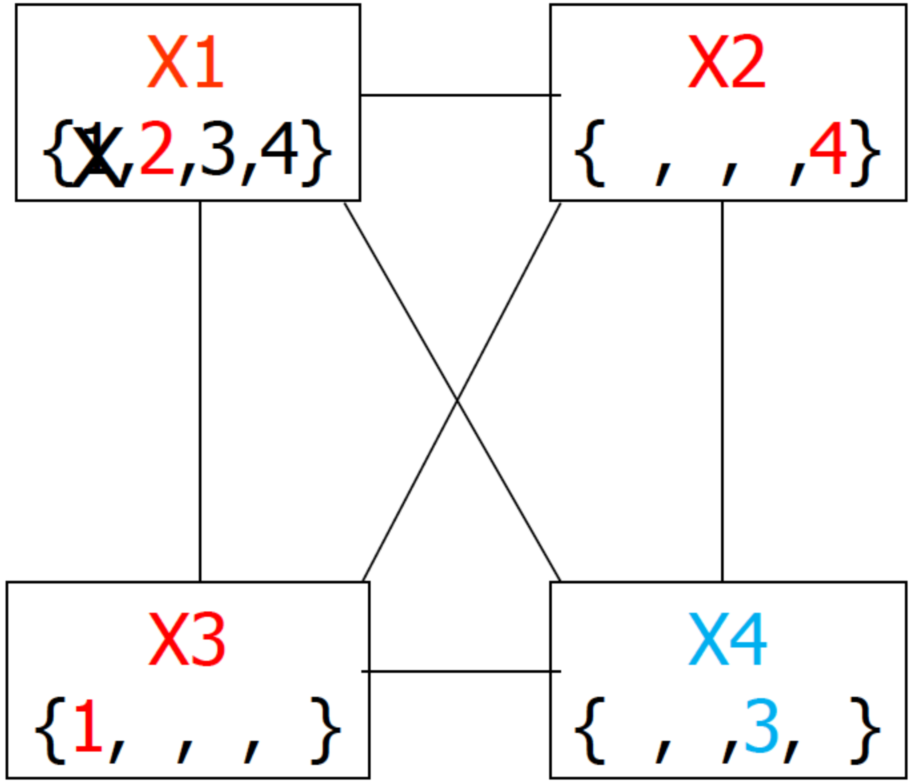




Red = value is assigned to variable
Blue = most recent variable/value pair
X = value led to failure

Forward Checking: N-Queens

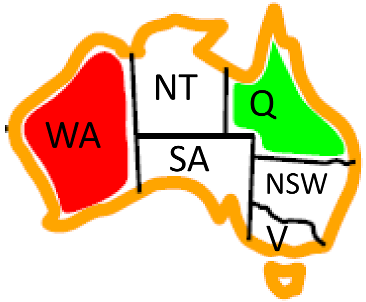




Red = value is assigned to variable
Blue = most recent variable/value pair
X = value led to failure

Forward Checking Limitations

- NT and SA cannot both be blue
- Forward checking does not recognize this
- Constraint propagation



Summary

- Optimization: we want to find a state that minimizes a cost function or maximizes a value function
- Local search methods make small changes to the state to search for an optimal configuration, often employing randomness
- Some of these optimization problems can easily be posed as constraint satisfaction problems, which allows us to exploit the domain-specific structure of the problem

Next Time

- Constraint propagation, selecting unassigned variables, ordering domain values