CS161 Discussion 5

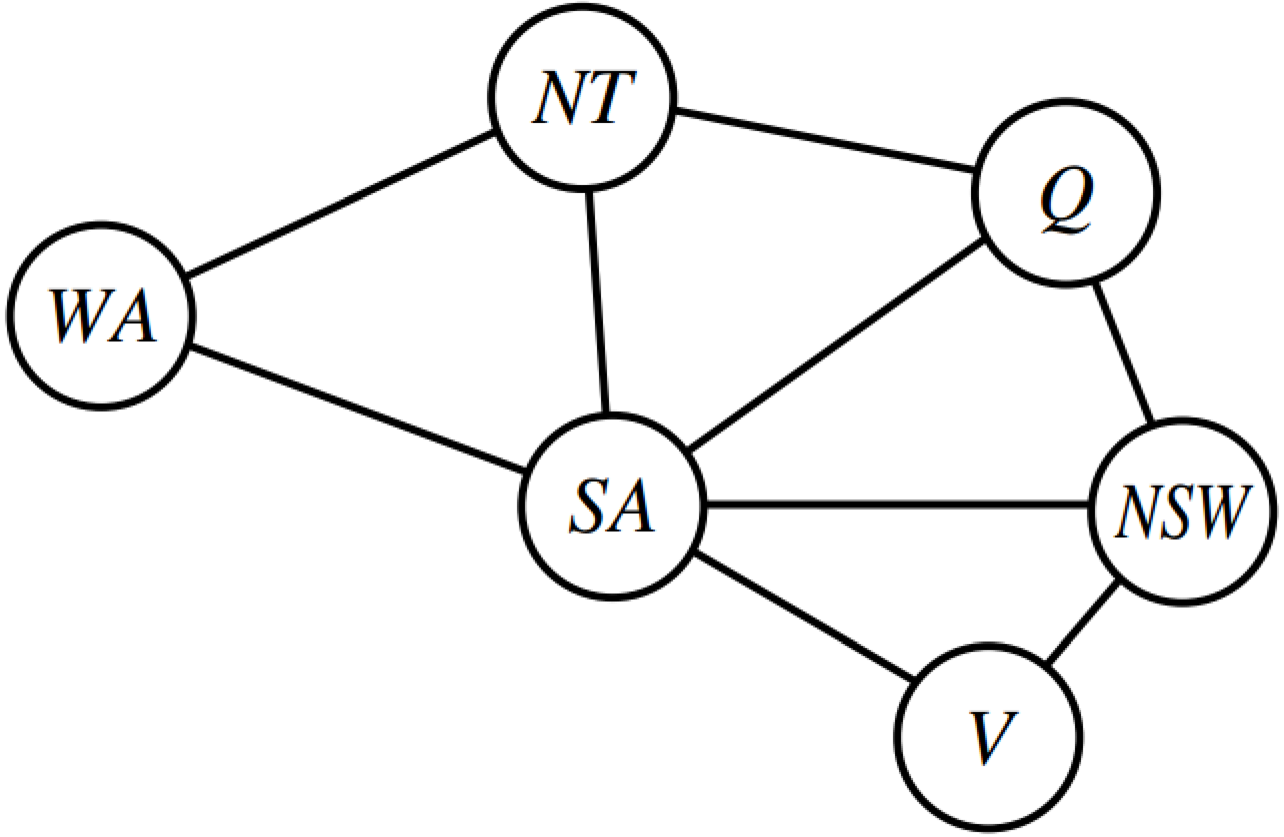
Shirley Chen

Constraint Satisfaction Problem

X is a set of variables, $\{X_1, \ldots, X_n\}$.
D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.
C is a set of constraints that specify allowable combinations of values.

- A **state** in CSP: an assignment of values to some or all variables
  - Consistent/Legal assignment: an assignment that does not violate any constraints
  - Complete assignment: every variable is assigned (otherwise partial assignment)
- A **solution** in CSP: a consistent, complete assignment

Constraint Graph



Exercise – CSP Formulations

- Class scheduling
  - A fixed number of professors
  - A fixed number of classrooms
  - A list of classes to be offered
  - A list of possible time slots for classes.
  - Each professor has a set of classes that he or she can teach.

Exercise – CSP Formulations

- Hamiltonian tour
  - Given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any.

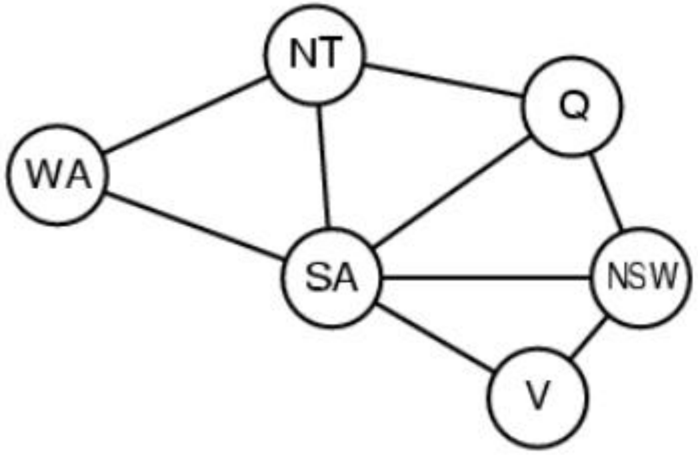
Backtracking DFS

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

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

Variable and Value Ordering

- How to select unassigned variable?
  - Minimum-remaining-values (MRV) heuristic
    - a.k.a. "most constrained variable", or "fail-first"
    - If no legal values left, fail immediately
  - Degree heuristic
    - Attempt to reduce branching factor on future choice
    - Useful as a tie-breaker



- In order what should its values be tried?
  - Least-constraining-value
    - Leave the maximum flexibility for subsequent variable assignments

Arc Consistency

- Is it possible to reduce variable domains **before** search?

- **Arc consistency**
  - Variable is arc consistent: Every value in its domain satisfies the variable's binary constraints
    - X_i is arc-consistent with respect to another variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)

  - Network is arc consistent: every variable is arc consistent with every other variable

Example – Arc Consistency

- $Y = X^2$
- The domain of both X and Y is the set of digits (0~9).

- Write it explicitly:
- <(X, Y), {(0, 0), (1, 1), (2, 4), (3, 9))}>

Exercise – Constraints Conversion

- Turn the ternary constraint "A+B=C" into three binary constraints. (Assume finite domains)

- Turn any ternary constraint into binary constraints.

- Eliminate unary constraints by altering domains of variables.

Conclusion:

Any CSP can be transformed into a CSP with only binary constraints.

Arc Consistency Algorithm: AC-3

- Maintains a queue (set) of arcs
- Pop an arbitrary arc (X_i, X_j)
    - D_i unchanged
        - Move to next
    - D_i becomes smaller
        - Add to queue all arcs(X_k, X_i) where X_k is a neighbor
    - D_i is empty
        - Fail!

Finally, we get an CSP that is equivalent to the original CSP(with same solutions).

But now variables have smaller domains!

Arc Consistency Algorithm: AC-3

```
function AC-3( csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE( csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Complexity of AC-3

- n variables
- d: largest domain size
- c binary constraints
- Each arc (X_k, X_i) can be inserted at most d times
  - Xi has at most d values to delete
- Checking consistency of one arc: $O(d^2)$

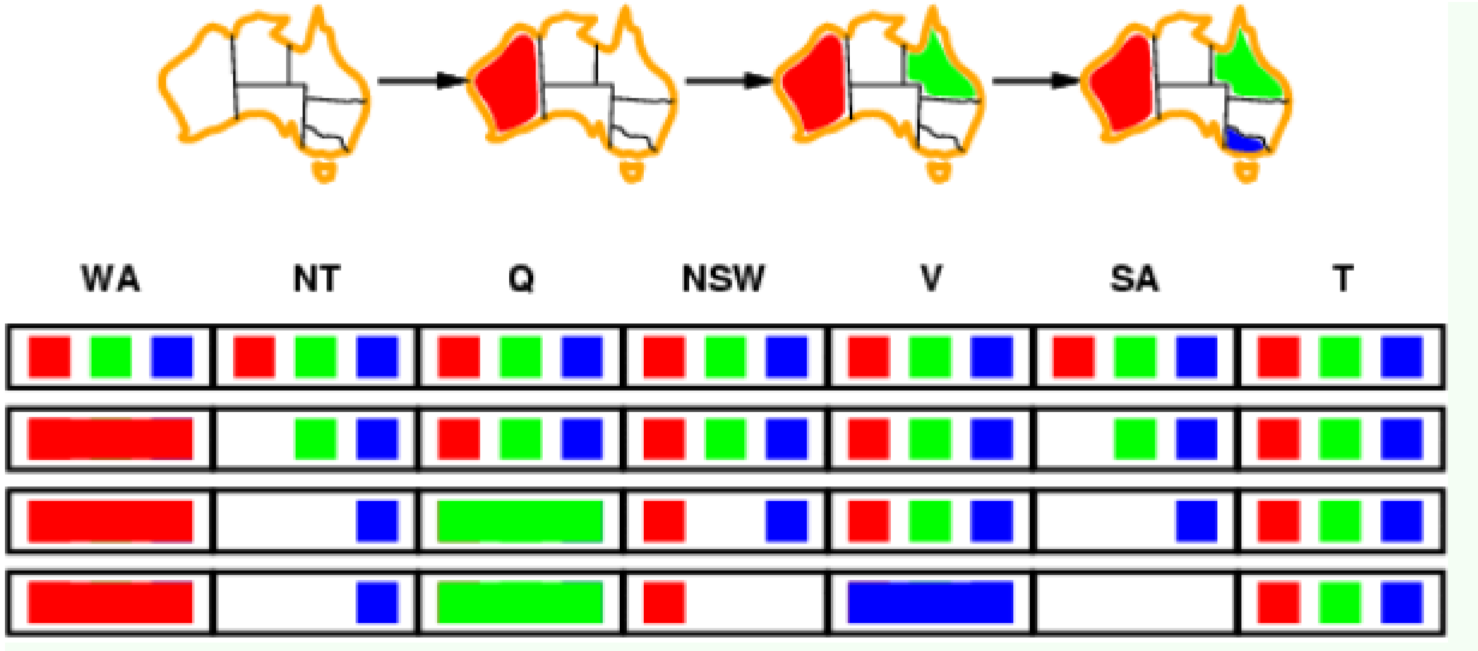
- $O(cd^3)$

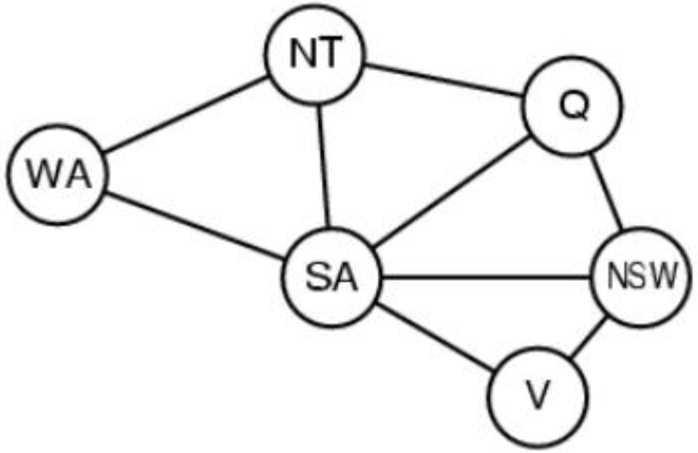
Forward Checking

- AC-3: infer domain reductions <u>before</u> search
- Can we do infer domain reductions in search?
  - And detect inevitable failure early

- Forward checking
  - Keep track of remaining legal values for unassigned variables that are connected to current variable. (Arc consistency)
  - Terminate search when any variable has no legal values

Example – Map Coloring





Maintaining Arc Consistency (MAC)

- Forward checking only makes current variable arc-consistent

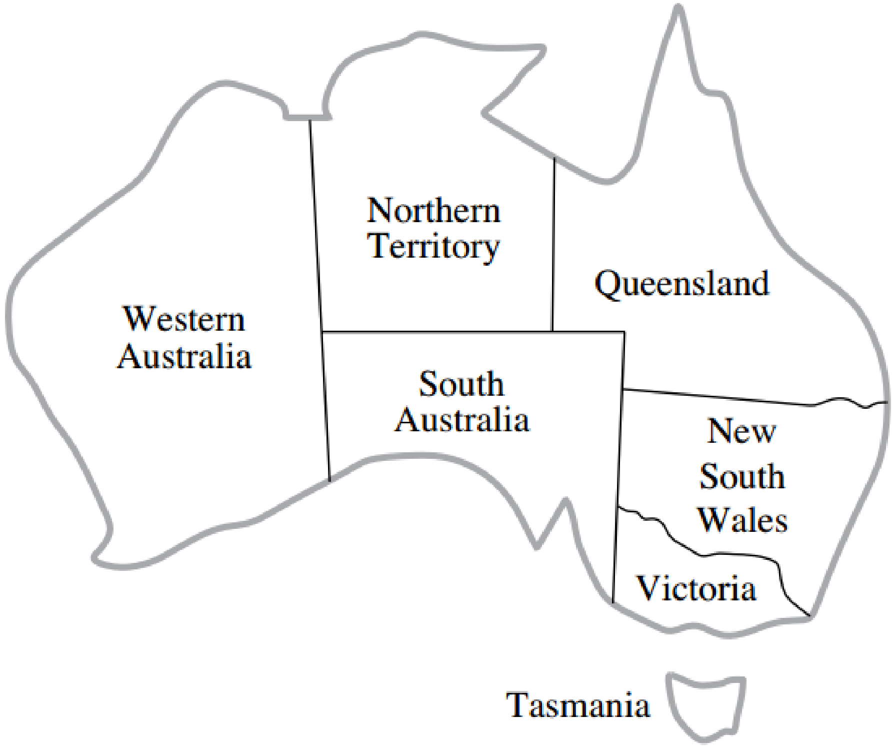
Maintaining Arc Consistency (MAC)

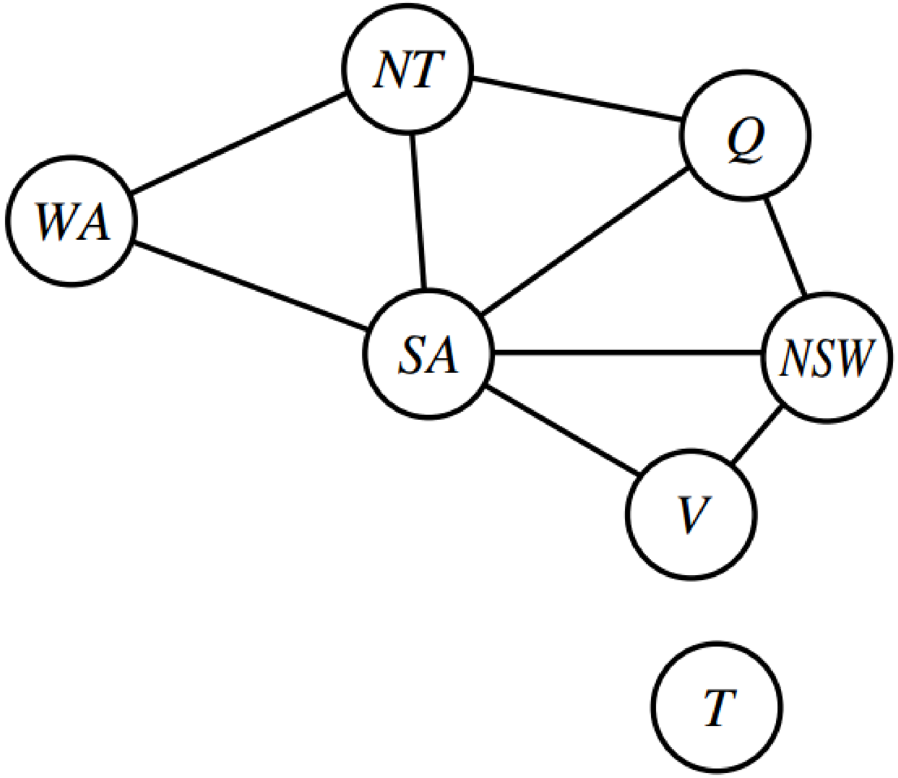
- After X_i is assigned a value
- Call AC-3 with (X_j, X_i) => constraint propagation
  - X_j: neighbor of X_i, unassigned

MAC is strictly more powerful than forward checking
Which one to pop? MRV

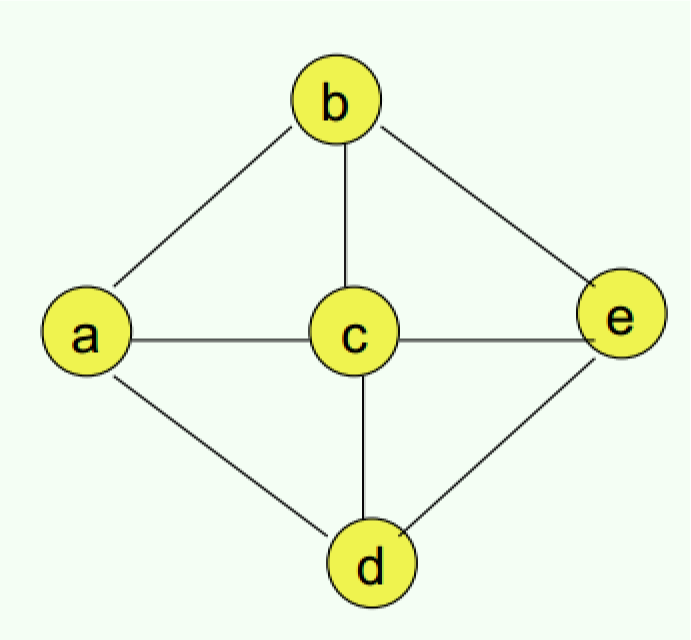
Exercise

- Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA = green, V = red}





Exercise – Solve CSP



- The domain for every variable is [1,2,3,4].
- 2 unary constraints:
  - variable “a” cannot take values 3 and 4.
  - variable “b” cannot take value 4.
- Variables connected by an edge cannot have the same value.

==========================

Heuristics: MRV, degree heuristic, forward checking

==========================

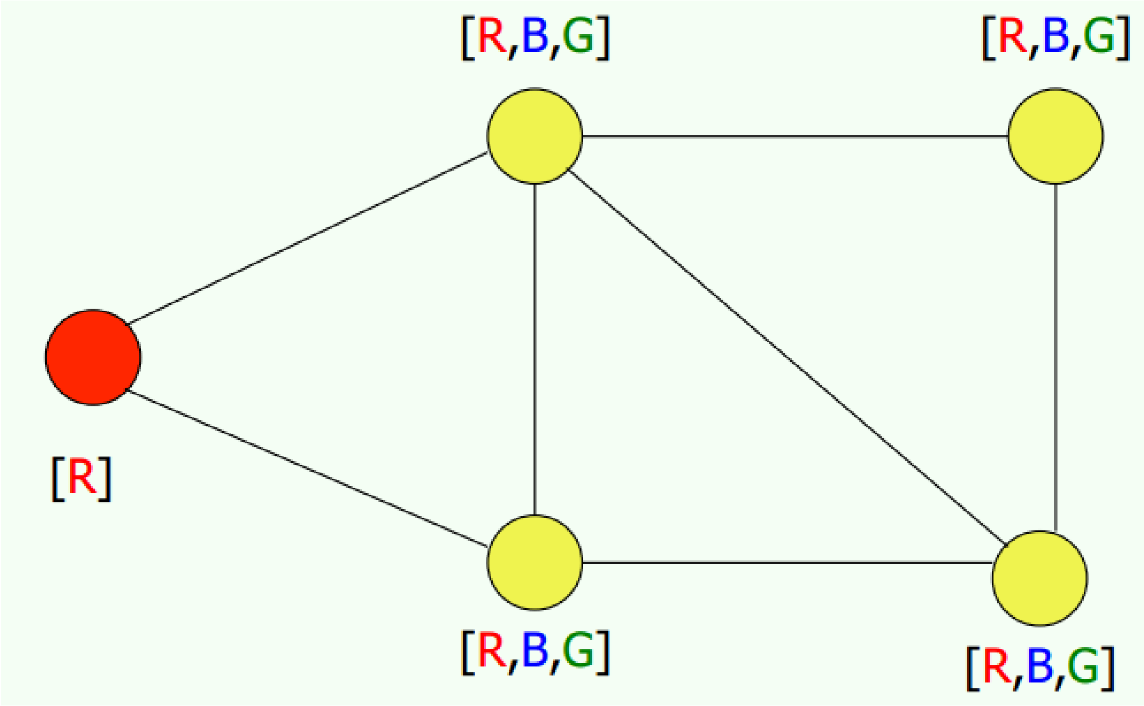
Find solution for this CSP. Show each step and explain.

Exercise – Solve CSP

- Connected variables cannot share color

Solve this CSP and explain each step

- Use all heuristics



Local Search

- Sometimes the path to the goal is irrelevant
- Only final configuration matters
  - n-queens, circuit design, road network,

Local Search

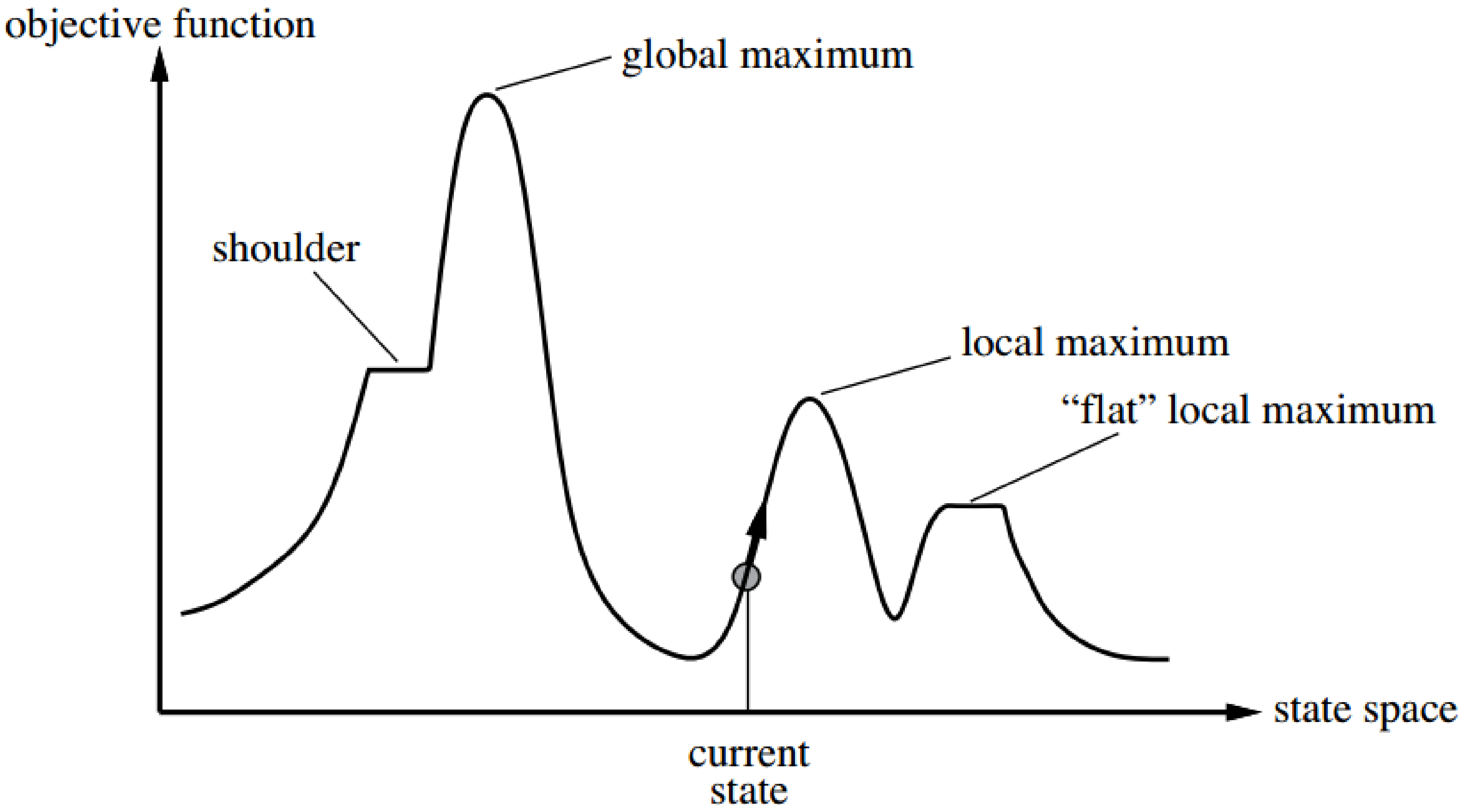
- Start from a single node
- Move to neighbors
- No need to keep paths

Local Search

Advantages:

- Little memory
- Find reasonable solution in large or infinite state spaces
  - Good for **optimization problems** (Find best state according to an **objective function**)

Optimization Problem

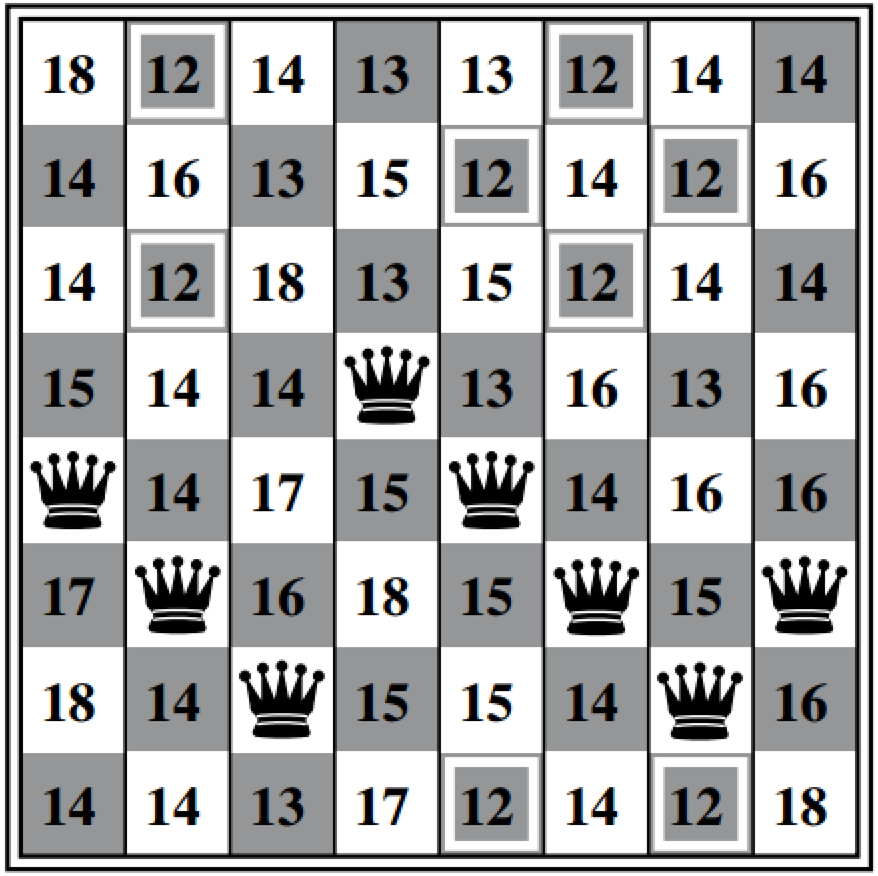


Hill-climbing search

- Greedy local search
    - Grabs a good neighbor state without thinking ahead about where to go next.

- Check all neighbors of current state
- choose the one with the highest value (lowest cost)
- Terminate when no neighbor has a higher value

Example – 8-queens

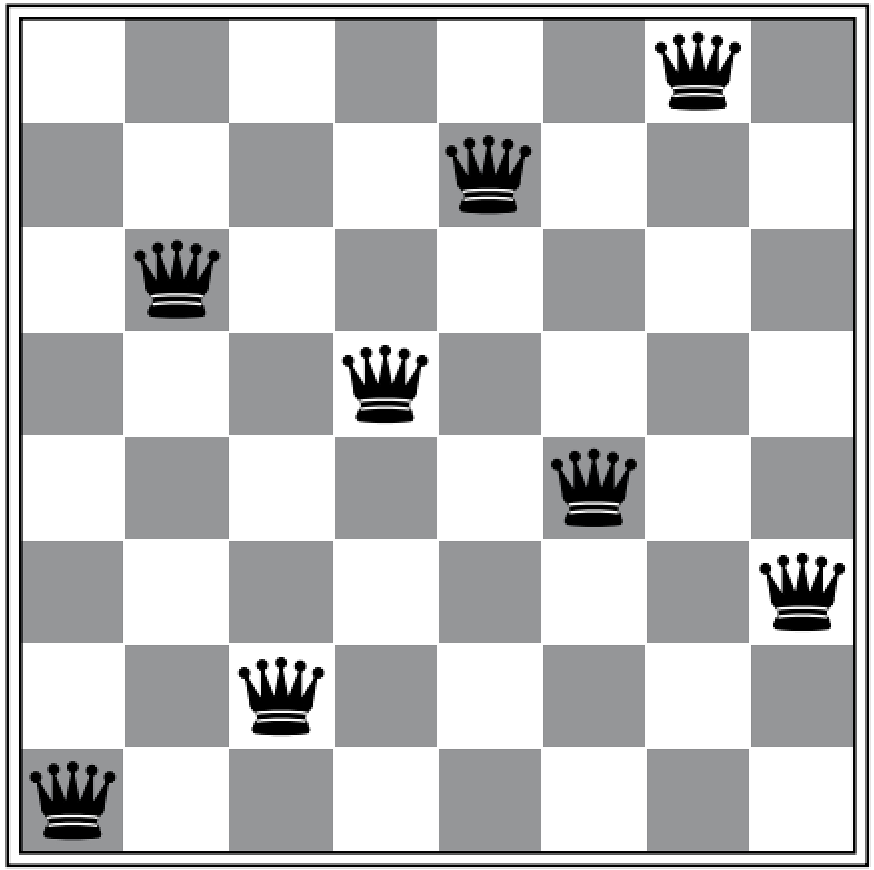


Hill-climbing search

- Advantage
  - Easy to improve a bad state (rapid progress)

- Disadvantage
  - Get stuck in
    - local optimal
    - Ridges
    - Plateau

Example – 8-queens



The state has h = 1 but every successor has a higher cost.

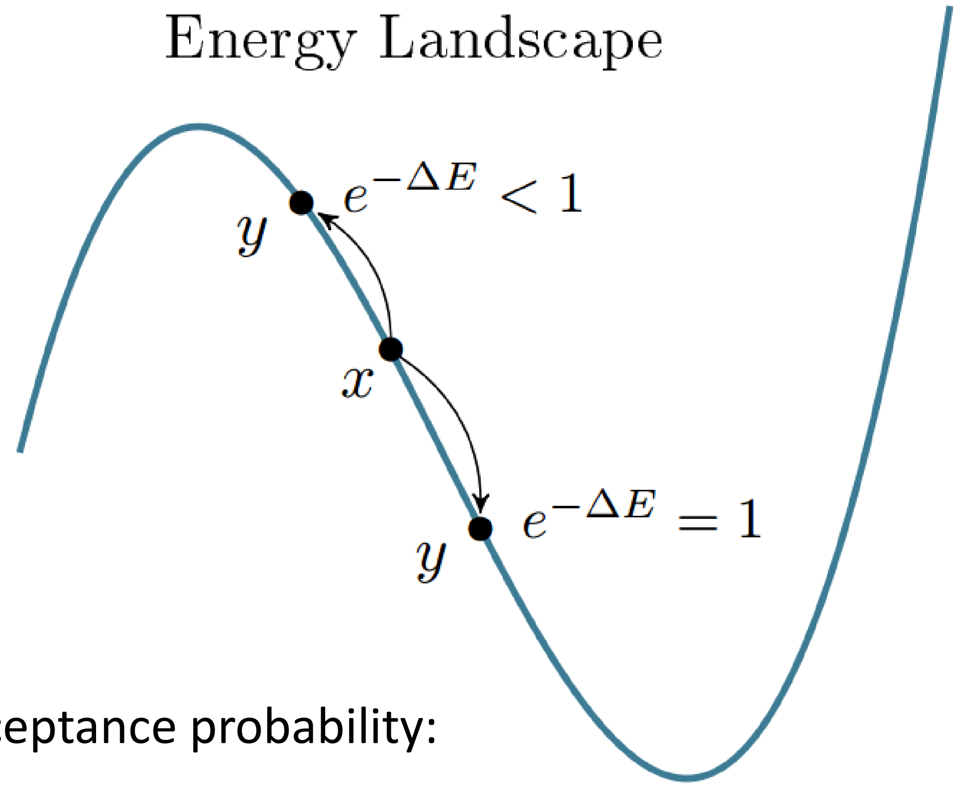
Hill-climbing search

- Disadvantage
    - Get stuck in
        - local optimal
        - Ridges
        - Plateau

The success of hill climbing depends very much on the shape of the state-space landscape!

NP-hard problems typically have an exponential number of local maxima to get stuck on.

Metropolis Methods



Acceptance probability:

$$\alpha(x, y) = 1, \quad \text{if } \Delta E < 0, \quad i.e.\ y \text{ is a lower energy state (better) than } x$$

$$\alpha(x, y) = e^{-\Delta E} < 1, \quad \text{if } \Delta E > 0, \quad i.e.\ y \text{ is a higher energy state (worse) than } x$$

Simulated Annealing

- Hill-climbing algorithms never move towards state with lower value
  - May result in local optimal

Simulated Annealing: an analogy of metropolis methods

- Randomly select candidate successor
- Go there if better
- Else go there with probability (Why?)
  function of "energy" and "temperature"

Example