Beyond Classical Search

CHAPTER 4
COSC 370
SPRING 2019
ALAN C. JAMIESON

SOME SLIDE CONTENT FROM RUSSELL & NORVIG PROVIDED SLIDES

1

- Hill-climbing
- Simulated annealing
- Local beam search
- Genetic algorithms
- Searching with non-determinism
- Searching with partial observations

2

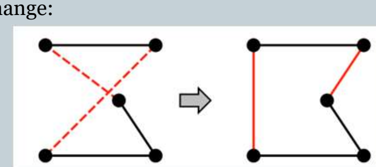
Local Searches

- Sometimes, we don't need to examine the whole state space (e.g. TSP).
- Iterative improvement algorithms – class of algorithms that focus on getting to the end state, rather than the path.
- State space = all possible complete configurations
- Optimize over those configurations!
- Constant space requirements.

3

Example: TSP

- Start with complete tour, perform pairwise exchange:

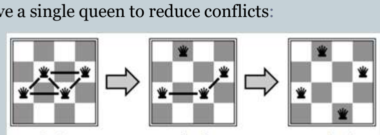


- Variants of this approach get near optimal solutions very quickly with thousands of cities.

4

Example: n-Queens

- Place n queens on an nxn board with no queen able to attack another.
- Strategy here?
  - Move a single queen to reduce conflicts:



- Very quick solution!

5

Hill-Climbing

- a.k.a. Gradient Ascent/Descent

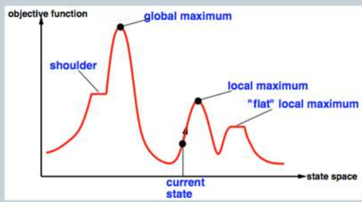
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

6

Watch the Landscape!

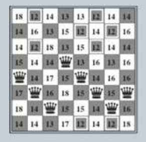
- Consider that the state space has a location and an elevation.
- Location = state itself
- Elevation = evaluation function result



7

Local Extremal States

- Local maximal/minimal states = bad news.



- How to get around? Random restart & random move.

8

Simulated Annealing

- Notice – hill-climbing techniques that never make a downhill move can be incomplete.
- Annealing – process used to temper or harden metals by heating them to high temperatures then gradually cooling them.
- Simulated annealing – gradient descent algorithm that incorporates some random moves to keep an agent out of local minimums/maximums.
- Allow "bad" moves but gradually decrease their frequency and size.

9

Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

10

Local Beam Search (briefly)

- Simulated Annealing = one state kept in memory.
- What if we were able to have *k* states? Would this speed things up?
- Local beam search – start with *k* randomly generated initial states and expand. Choose the *k* best evaluated nodes expanded and continue until you get the goal.
- Stochastic beam search expands this further by utilizing a probability to choose *k* random expanded states.

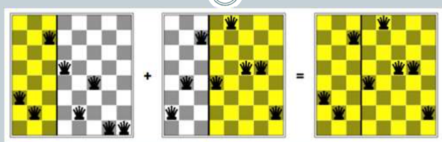
11

Genetic Algorithms (briefly)

- A expansion of the stochastic beam search.
- Choose population (randomly selected states).
- Each individual state must be represented by a string over a finite alphabet (binary bits, characters, etc.)
  - For instance: n-queens states represent by bits the location of each queen, 8-queens needs 24 bits, 3 bits per column.
- The next generation of states is created by examining the current set of states and their fitness (typically by the standard evaluative function). Then a random (but probabilistically chosen) set of states are chosen. Those states are paired and a crossover point is chosen randomly. New states are created from the two parents by appending substrings from the parents together based on the crossover.

12

Genetic Algorithms (briefly)



- Mutations can be added in with small independent probability. This will be dependent on the problem, but represent a change in a generated state's string. For instance – moving a single queen randomly in its column.

13

Adding Nondeterminism

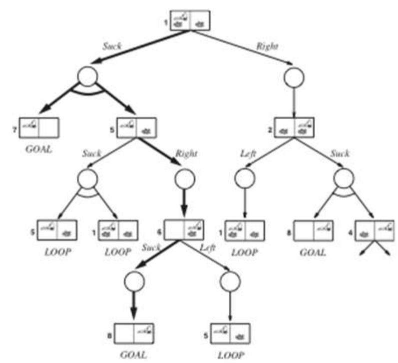
- Consider an erratic Vacuum World - when the *Suck* command is executed the vacuum cleans just the current square, cleans the currents square and an adjacent square, or deposits dirt onto a clean square.
- How do we deal with this non-determinism? What does our state space look like? How do we search when we don't know what's next?
- Consider all the options!

14

AND-OR Search Trees

- Search trees where certain branches represent decisions (OR) and certain branches represent options that must all be taken.
- For instance: our *Suck* command on a clean square.
- What about just sitting at a particular state?
- We assume that we have a fully observable environment.

15



16

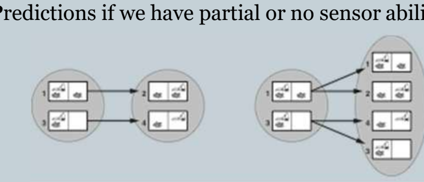
What about…

- A slippery Vacuum World? Movement actions occasionally fail.
- Try, try again – an extension of our *if-else* state based operations before, occasionally we will have to loop back to the state where the failure occurred, then continue to execute as normal.
- Requires some knowledge of whether or not an action failed.

17

Partial/No Observations (briefly)

- How do we deal with an agent that doesn't provide enough information to set state?
- Belief states: a set of fully observable states representing all possible configurations of state.
- Predictions if we have partial or no sensor ability!



(a) (b)

18

Quick Exercise!

- 4.1 a-d

19