CSCI 360
Introduction to Artificial Intelligence

Instructor: Wei-Min Shen
Today’s Lecture

- Search Techniques (review & continue)
- Optimization Techniques
- Home Work 1: description and assignment
Search Techniques (Review)

• Breadth-first
• Depth-first
• Best-first
  – A* (past + admissible future)
    • Never overestimate the future
  – Dynamic Programming (past + V(s) to goal)
    • Not path by path, but stage to stage
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first

• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?

• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

• HM: state space for TOH, assign values for state and actions.
Time complexity of depth-first

• In the worst case:
  • the (only) goal node may be on the right-most branch,
  
  \[
  \text{Time complexity} = b^d + b^{d-1} + \ldots + 1 = \frac{(b^{d+1} - 1)}{(b-1)}
  \]
  
  Thus: \( O(b^d) \)
  
  **What if \( d = \infty \)?"
Limited/Combined “Search Horizons”

• “Iterative Deepening” method
  – Limit the search depth, incrementally increase it

• “Bi-Directional” Search
  – Search from the initial to the goal, as well as from the goal to the initial
Solving Problems by Search

• How to represent a problem?
  – Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

• How to solve a problem?
  – Search: from here to there, from the initials to the goals
  – Depth-first, breadth-first

• How good is your solution? (fig 6.1, ALFE)
  – How good is your state? How costly is an action?
  – Best-first, Dynamic Programming, A*, etc.
  – Can you guarantee anything? (optimal vs heuristic)

• How much do you want to pay for your solution?
  – How deep/wide can you go?
    • Predetermined vs dynamic (e.g., iterative deepening)
  – One way or many ways (bi-directional)?

• How big is a problem? Can you put the whole world in your head?
  – Tower of Hanoi, chess, robot-and-world,

• HM: state space for TOH, assign values for state and actions.
How Big is a problem?

- 8-puzzle has 362,880 states
- 15-puzzle has $10^{12}$ states
- 24-puzzle has $10^{25}$ states

When you solve $N=64$, it would be the end of the world!
Time complexity of depth-first

- In the worst case:
  - the (only) goal node may be on the right-most branch,

Time complexity = $b^d + b^{d-1} + \ldots + 1 = (b^{d+1} - 1)/(b-1)$

Thus: $O(b^d)$

What if $d = \infty$ ?
Solving Problems by Search

- How to represent a problem?
  - Examples (e.g., 8puzzle, TOH, Roomba, Travel), states, actions, initials, goals.

- How to solve a problem?
  - Search: from here to there, from the initials to the goals
  - Depth-first, breadth-first

- How good is your solution? (fig 6.1, ALFE)
  - How good is your state? How costly is an action?
  - Best-first, Dynamic Programming, A*, etc.
  - Can you guarantee anything? (optimal vs heuristic)

- How much do you want to pay for your solution?
  - How deep/wide can you go?
    - Predetermined vs dynamic (e.g., iterative deepening)
  - One way or many ways (bi-directional)?

- How big is a problem? Can you put the whole world in your head?
  - Tower of Hanoi, chess, robot-and-world,

- HM: state space for TOH, assign values for state and actions.
A Homework Problem

• Tower of Hanio (n=3)
  – 1. Write the state space, states, actions
  – 2. Assume the cost of an action to move a disk is the weight of the disk, compute the “best future cost” $V(s_0)$ for the start state using dynamic programming

Start state $s_0$

The weight of the disk: disk1=1, disk2=2, disk3=3
Optimization

- What is the optimization?
- Why is it so important?
- Why is it so hard?
- How do we do it today?
What is the Problem? So “simple”?

- Given: an objective function $F(x)$
- Find: a $x^*$ such that $F(x^*)$ is the global extreme
  $F(x^*) = \max \{ F(x) \}$ or $F(x^*) = \min \{ F(x) \}$
Why is it so important?

• All engineering problems are optimizations!
  – Design: architecture, software, robots, website, ...
  – AI systems: industry, agriculture, military, finance, ...
  – For yourself: Getting good grades 😊

• Represent your desires as the objective function
  – Choice 1: $x$ as a state, $F(x)$ rewards the better states
  – Choice 2: $x$ as a path, $F(x)$ rewards the better paths
  – Degree of “goodness”: goal related, cost related, etc.
  – Let us look at some examples
A Challenging Claim!

• All engineering problems are optimization
  – All or Almost All 😊
  – We still don’t have a general solution for it!

• The key: Represent them properly

• “Prove” this claim by yourself
  – Examples
The Cleaning Robot

- $x$: 00, 01, 10, 11, ( 
  - State: LeftRoom_RightRoom, (dirty=0, clean=1)
- $F(x) = x$; “the cleaner the room, the higher $F(x)$”
Travel in Romania

• Choice 1: x as a city
  – F(x) is the distance from x to Budapest

• Choice 2: x is a sequence of cities starting from Arad
  – F(x) has a higher value if the path ends at Budapest
  – What about “leads to Budapest”? What about the “cost”?
Homework: design $x$ and $F(x)$ for these so that they become an optimization problem.
Project-1 as an Optimization Problem

Representation: $x$ is the current location of the robot.
$F(x)$ is the distance from the robot to the goal.
Question: What about the orientation of the robot?
Dynamic Programming Example

Note: x is a state, but F(x) is about the most rewarding path from x to a goal state.

\[
V(s_1) = \max\{R(s_1, a_1) + V(s_4), R(s_1, a_2) + V(s_5)\} = \max\{1 + 2, 3 + 1\} = 4
\]

\[
V(s_2) = \max\{R(s_2, a_0) + V(s_4), R(s_2, a_1) + V(s_5), R(s_2, a_2) + V(s_6)\}
= \max\{6 + 2, 2 + 1, 5 + 1\} = 8
\]

\[
V(s_3) = \max\{R(s_3, a_0) + V(s_5), R(s_3, a_1) + V(s_6)\} = \max\{2 + 1, 3 + 1\} = 4
\]

\[
V(s_i) = \max_{a} \{R(s_i, a) + V(s_j)\} \quad \text{where } s_i \xrightarrow{a} s_j
\]
Why is it so hard?

- Why is it so hard? (no silver-bullet solution)
  - Which way to go next?
  - How much can you see? (local/partial vs global/complete sensors)
  - How many points can you remember (incremental)?
  - How small is your step? (not to skip x*)
  - How to get from one x to another (bound by actions)?
  - How well can you guess (the next x)?
  - What do you know about the function (continuous)?
  - How to check if you are done (avoid local extremes)?
  - Will the function F(x) change by itself (often does)?
  - How to design the objective function?
    • In AI, we call it the Representation Challenge

- We still don’t have a general solution for all
- We will illustrate all these using the following picture
Why is it so hard?
Representation Challenge for AI

• Why is “representation” so important?
• A great example from Herbert A. Simon
  – Game: Make a book of 15 from 9 cards
  – Goal: first does wins (can you always win?)
  – “The Science of the Artificial” p131
• This is still an open challenge for AI
Some Existing Approaches
(Discuss the details in class)

- Dynamic programming
- Hill climbing
  - Idea: Use local gradient(x) to determine direction, always heads to the better
  - Pros: simple, local, incremental, no memory
  - Cons: may be trapped in local extreme
- Simulated Annealing
  - Ideas: long and random jumps when temperature is high
  - Pros: may avoid local extreme
  - Cons: expensive, not always find $x^*$,
- Genetic algorithms
- Sampling techniques (e.g., random walks)
- Online (incremental) search
- Non-stationary search techniques
- Many more “new” methods are being invented as we speak
- For Project 1: What are you using, if any?
Hill climbing (or gradient ascent/dist下降)

- Iteratively maximize "value" of current state, by replacing it by successor state that has highest value, as long as possible.

"Like climbing Everest in thick fog with amnesia"

```plaintext
function HILL-CLIMBING(problem) returns a solution state
    inputs: problem, a problem
    local variables: current, a node
                    next, a node

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

• Note: minimizing a “value” function $v(n)$ is equivalent to maximizing $-v(n)$,

thus both notions are used interchangeably.

• Notion of “extremization”: find extrema (minima or maxima) of a value function.
Hill climbing

- **Problem:** depending on initial state, may get stuck in local extremum.
Minimizing energy

• Let’s now change the formulation of the problem a bit, so that we can employ new formalism:
  – let’s compare our state space to that of a physical system that is subject to natural interactions,
  – and let’s compare our value function to the overall potential energy $E$ of the system

• On every updating, we have $\Delta E \leq 0$
• Hence the dynamics of the system tend to move $E$ toward a minimum
• Note that there may be different such states — they are *local* minima
• Global minimization is not guaranteed.
Local Minima Problem

• Question: How do you avoid this local minimum?
Consequences of the Occasional Ascents

desired effect

Help escaping the local optima.

adverse effect

Might pass global optima after reaching it

(easy to avoid by keeping track of best-ever state)
Boltzmann Machines

- To motivate their solution, consider how one might get a ball-bearing traveling along the curve to "probably end up" in the deepest minimum.
- The idea is to shake the box "about h hard" — then the ball is more likely to go from D to C than from C to D. So, on average, the ball should end up in C's valley.
Simulated annealing: basic idea

- From current state, pick a random successor state;
- If it has better value than current state, then “accept the transition,” that is, use successor state as current state;
- Otherwise, do not give up, but instead flip a coin and accept the transition with a given probability (that is lower as the successor is worse).
- So we accept to sometimes “un-optimize” the value function a little with a non-zero probability.
Boltzmann’s statistical theory of gases

- In the statistical theory of gases, the gas is described not by a deterministic dynamics, but rather by the probability that it will be in different states.

- The 19th century physicist Ludwig Boltzmann developed a theory that included a probability distribution of temperature (i.e., every small region of the gas had the same kinetic energy).

- Hinton, Sejnowski and Ackley’s idea was that this distribution might also be used to describe neural interactions, where low temperature $T$ is replaced by a small noise term $T$ (the neural analog of random thermal motion of molecules). While their results primarily concern optimization using neural networks, the idea is more general.
Boltzmann distribution

- At thermal equilibrium at temperature $T$, the **Boltzmann distribution** gives the relative probability that the system will occupy state $A$ vs. state $B$ as:

\[
\frac{P(A)}{P(B)} = \exp\left(-\frac{E(A) - E(B)}{T}\right) = \frac{\exp(E(B)/T)}{\exp(E(A)/T)}
\]

- where $E(A)$ and $E(B)$ are the energies associated with states $A$ and $B$. 
Simulated Annealing

Kirkpatrick et al. 1983:

• **Simulated annealing** is a general method for making likely the escape from local minima by allowing jumps to higher energy states.

• The analogy here is with the process of annealing used by a craftsman in forging a sword from an alloy.

• He heats the metal, then slowly cools it as he hammers the blade into shape.
  
  – If he cools the blade too quickly the metal will form patches of different composition;
  
  – If the metal is cooled slowly while it is shaped, the constituent metals will form a uniform alloy.
Real Annealing: Sword

- He heats the metal, then slowly cools it as he hammers the blade into shape.
  - If he cools the blade too quickly the metal will form patches of different composition;
  - If the metal is cooled slowly while it is shaped, the constituent metals will form a uniform alloy.
Simulated annealing in practice

- set T
- optimize for given T
- lower T
- repeat

(see Geman & Geman, 1984)
Simulated annealing in practice

- set $T$
- optimize for given $T$
- lower $T$
- repeat

MDSA: Molecular Dynamics Simulated Annealing
Simulated annealing in practice

- set T
- optimize for given T
- lower T
- repeat

(see Geman & Geman, 1984)

• Geman & Geman (1984): if T is lowered sufficiently slowly (with respect to the number of iterations used to optimize at a given T), then simulated annealing is guaranteed to find the global minimum.

• Caveat: this algorithm has no end (Geman & Geman’s T decrease schedule is in the 1/log of the number of iterations, so, T will never reach zero), so it may take an infinite amount of time for it to find the global minimum.
Simulated annealing algorithm

- Idea: Escape local extrema by allowing “bad moves,” but gradually decrease their size and frequency.

```plaintext
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 the probability 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}

Note:
   Δ E>0 to maximize E, or
   Δ E<0 to minimize E
```
Note on simulated annealing: limit cases

- **Boltzmann distribution:** accept “bad move” with $\Delta E < 0$ (goal is to maximize $E$) with probability $P(\Delta E) = \exp(\Delta E/T)$

- If $T$ is large: $\Delta E < 0$
  $\Delta E/T < 0$ and very small
  $\exp(\Delta E/T)$ close to 1
  accept bad move with **high** probability

- If $T$ is near 0: $\Delta E < 0$
  $\Delta E/T < 0$ and very large
  $\exp(\Delta E/T)$ close to 0
  accept bad move with **low** probability
Some Existing Approaches
(Discuss the details in class)

• Dynamic programming
• Hill climbing
  – Idea: Use local gradient(x) to determine direction, always heads to the better
  – Pros: simple, local, incremental, no memory
  – Cons: may be trapped in local extreme
• Simulated Annealing
  – Ideas: long and random jumps when temperature is high
  – Pros: may avoid local extreme
  – Cons: expensive, not always find $x^*$,
• Genetic algorithms
• Sampling techniques (e.g., random walks)
• Online (incremental) search
• Non-stationary search techniques
• Many more “new” methods are being invented as we speak
• For Project 1: What are you using, if any?
Summary of Search/Optimization

**Uninformed:** Use only information available in the problem formulation (Uniform-cost)
- Breadth-first
- Depth-first
- Depth-limited
- Iterative deepening

**Informed:** Use knowledge or heuristics to guide the search ("best" first)
- Dynamic programming: iteratively compute state values, choose the best state
- Greedy search – expand the node that has the maximal value
- A* search – expand the node maximize “past + permissive future”.
- Iterative improvement – keep no memory of path; work on a single current state and iteratively improve its “value.”
- Hill climbing – select the next state which maximizes value.
- Simulated annealing – refinement on hill climbing by which “bad moves” are permitted, but with decreasing size and frequency. Overcome local minimal