Today 9/12
• Review informed searches
• Start on local, iterative improvement search

Review
• How is the agenda ordered in the following searches?
  - Uniform Cost
  - Best First
  - A*
  - IDA*
Review: $A^*$ search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n) =$ cost so far to reach $n$
- $h(n) =$ estimated cost from $n$ to goal
- $f(n) =$ estimated total cost of path through $n$ to goal

$A^*$ search example

$A^*$ search example
A* search example
A* search example

Remaining Search Types

- Backtracking state-space search
- Optimization search
- Constraint satisfaction search

Optimization

- Sometimes referred to as iterative improvement or local search.
- We'll talk about three simple but effective techniques:
  - Hillclimbing
  - Random Restart Hillclimbing
  - Simulated Annealing
Optimization Framework

- Working with 1 state in memory
  - No agenda/queue/fringe...
  - Usually
- Usually generating new states from this 1 state in an attempt to improve things
- Goal notion is slightly different
  - Normally solutions are easy to find
  - We can compare solutions and say one is better than another
  - Goal is usually an optimization of some function of the "solution" (cost).

Numerical Optimization

- We're not going to consider numerical optimization approaches...
- The approaches we're considering here don't have well-defined objective functions that can be used to do traditional optimization.
- But the techniques used are related

Hill-climbing Search

- Generate nearby successor states to the current state based on some knowledge of the problem.
- Pick the best of the bunch and replace the current state with that one.
- Loop (until?)
Hill-Climbing Search

function HILL-CLIMBING(problem) return a state that is a local maximum
input: 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

Hill-climbing

• Implicit in this scheme is the notion of a neighborhood that in some way preserves the cost behavior of the solution space...
  - Think about the TSP problem again
  - If I have a current tour what would a neighboring tour look like?
    • This is a way of asking for a successor function.

Hill-climbing Search

• The successor function is where the intelligence lies in hill-climbing search
• It has to be conservative enough to preserve significant “good” portions of the current solution
• And liberal enough to allow the state space to be preserved without degenerating into a random walk
Hill-climbing search

- Problem: depending on initial state, can get stuck in various ways

Break

- Questions?
- Python problems?
- My office hours are now
  - Tuesday 2 to 3:30
  - Thursday 12:30 to 2
- Go to cua.colorado.edu to view lectures (Windows and IE only)

Quiz Alert

- The first quiz is on 9/21 (A week from Thursday)
- It will cover Chapters 3 to 6
  - I’ll post a list of sections to pay close attention to
  - I’ll post some past quizzes soon (remind me by email)
Local Maxima (Minima)

• Hill-climbing is subject to getting stuck in a variety of local conditions...
• Two solutions
  - Random restart hill-climbing
  - Simulated annealing

Random Restart Hillclimbing

• Pretty obvious what this is....
  - Generate a random start state
  - Run hill-climbing and store answer
  - Iterate, keeping the current best answer as you go
  - Stopping... when?
• Give me an optimality proof for it.

Annealing

• Based on a metallurgical metaphor
  - Start with a temperature set very high and slowly reduce it.
  - Run hillclimbing with the twist that you can occasionally replace the current state with a worse state based on the current temperature and how much worse the new state is.
Annealing

- More formally...
  - Generate a new neighbor from current state.
  - If it's better take it.
  - If it's worse then take it with some probability proportional to the temperature and the delta between the new and old states.

Simulated annealing

```plaintext
function SIMULATED-ANNEALING(problem, schedule) return a solution state

input: 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
  ∆E ← VALUE[next] - VALUE[current]
  if ∆E > 0 then current ← next
  else current ← next only with probability exp(∆E / T)
```

Properties of simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Widely used in VLSI layout, airline scheduling, etc
Coming Up

- Thursday: Constraint satisfaction (Chapter 5)
- Tuesday: Game playing (Chapter 6)
- Thursday: Quiz