

Optimization

Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

1

Announcements

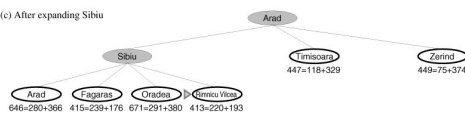
- Project groups and preliminary topic ideas will be due on 6/30
 - A week from Monday
 - Be thinking about what you'd like to do
 - Try to find others in the class who might be interested in the same topic!
- We're almost ready to start using the class discussions on the mailing list

2

Last Time

- Someone asked why, in the textbook's example on page 98, A* search looks backward to nodes already explored
- The answer is: they don't appear to be using a closed list
- If $g(n) \geq 0$ for all n , closed lists prevent such unnecessary work (back-tracked states will always be toward the end of the queue)
- However, if $-\infty \leq g(n) \leq \infty$ (i.e. costs can be negative: or rewards!), you *would* want to allow such moves, and ignore the closed list
- Closed lists aren't necessary, but are often useful

(c) After expanding Sibiu



3

Searching: So Far

- We've discussed how to build goal-based and utility-based agents that *search* to solve problems
- We've also presented both uninformed (or *blind*) and informed (or *heuristic*) approaches for search
- What we've covered so far are called partial search strategies because they build up partial solutions, which could enumerate the *entire* state space before finding a solution

4

Complete Searching

- In complete search strategies, each state or node already represents a complete solution to the problem at hand
 - We aren't concerned with finding a *path*
 - We don't necessarily have a *designated* start state
- The objective is to search through the problem space to find other solutions that are better, the best, or that that meet certain criteria (goal)

5

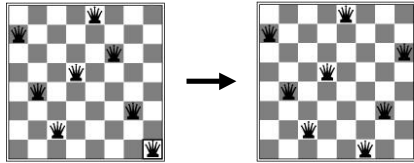
Optimization

- Problems where we search through complete solutions to find the *best* solution are often referred to as optimization problems
- Most optimization tasks belong to a class of computational problems called NP
 - Non-deterministic Polynomial time solvable
 - Computationally very hard problems
 - For NP problems, state spaces are usually exponential, so partial search methods aren't time or space efficient

6

Optimization Problems

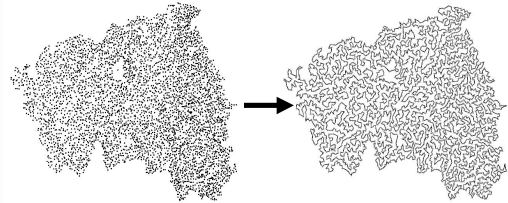
- The k -Queens Problem
Of course this isn't *real* chess



7

Optimization Problems

- Traveling Salesman Problem (TSP)
Perhaps most famous optimization problem



8

Optimization Problems

- As it turns out, many real-world problems that we might want an agent to solve are similarly hard optimization problems:
 - Bin-packing
 - Logistics planning
 - VLSI layout/circuit design
 - Theorem-proving
 - Navigation/routing
 - Production scheduling, supply/demand
 - Learning the parameters for a neural network (more in the machine learning part of the course)

9

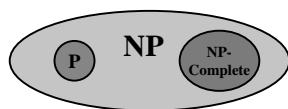
Optimization Problems

- For optimization (also sometimes called constraint-satisfaction) problems, there is a well-defined objective function that we are trying to optimize

10

Satisfiability (SAT)

- Classic NP problem
- Belongs to a specific class of problems in the complexity hierarchy called NP-Complete
 - Any other NP problem can be converted to a SAT problem in polynomial time
 - These are the hardest problems we know of



11

Satisfiability (SAT)

- Given:
 - Some logical formula
 - Array of binary variables in the formula
- Do:
 - Find a truth assignment for all variables such that the formula is *satisfied* (true)

12

3

Greedy Search for SAT

- Greedy search does the right thing in that it does find a solution, and quickly
- However, it only expanded 4 out of the 35 that are generated in the search (*i.e.* placed in the open list)
 - You may work it all out yourself if you wish
- It also found a direct route, and we don't need to remember the path, so storing all those extra states pretty much wasted space!

19

Local Search

- Local search is a type of greedy, complete search that focuses on a specific (or *local*) part of the search space, rather than trying to branch out into all of it
- We only consider the neighborhood of the *current state* rather than the entire state space so far (so as not to waste time/space)

20

Beam Search

- One type of local search is beam search, which uses $f(n)$, as in other informed searches, but uses a “beam” with a width w to restrict the possible search directions
- Only keep the w -best nodes in the open list, and throw the rest away
- More space efficient than best-first search, but can throw away nodes on a solution path

21

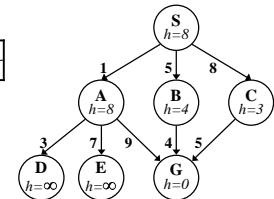
Beam Search Example

$$f(n) = g(n) + h(n)$$

$$w = 2$$

of states tested: 0, expanded: 0

State	Open	Closed
...	S:8	..



22

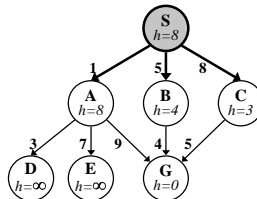
Beam Search Example

$$f(n) = g(n) + h(n)$$

$$w = 2$$

of states tested: 1, expanded: 1

State	Open	Closed
...	S:8	..
S	A:9, B:9, C:11	S



23

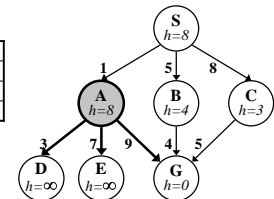
Beam Search Example

$$f(n) = g(n) + h(n)$$

$$w = 2$$

of states tested: 2, expanded: 2

State	Open	Closed
...	S:8	..
S	A:9, B:9	S
A	B:9, G:10, D:∞, E:∞	SA



24

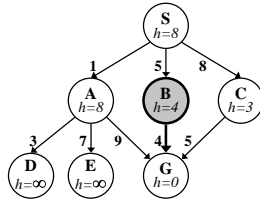
Beam Search Example

$$f(n) = g(n) + h(n)$$

$w = 2$

of states tested: 3, expanded: 3

State	Open	Closed
..	S:8	..
S	A:9, B:9	S
A	B:9, G:10	SA
B	G:9	SAB



25

Beam Search Example

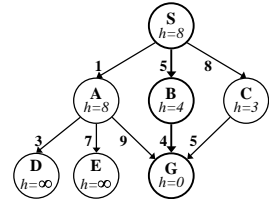
$$f(n) = g(n) + h(n)$$

$w = 2$

of states tested: 4, expanded: 3

State	Open	Closed
..	S:8	..
S	A:9, B:9	S
A	B:9, G:10	SA
B	G:9	SAB
G	Goal!!	

Space used (Beam): 4
Space used (A*): 7



26

Hill-Climbing (HC)

- The most common local strategy is called hill-climbing, if the task is to maximize the objective function
 - Called gradient descent if we are minimizing
- We consider all the successors of the current node, expand the best one, and throw the rest away

27

Hill-Climbing (HC)

- HC chooses what looks best locally like greedy search, but cannot backtrack or consider an alternative path (there is no open list)
- HC is very simple and space efficient
 - Like beam search with $w = 1$
 - However, a closed list can become very large if used (but it usually isn't)

28

Hill-Climbing (HC)

```

CURRENT = initialState // initialize the search
loop forever { // loop until optimum is found
    NEXT = highest-valued successor of CURRENT
    if score(CURRENT) better than score(NEXT) then
        return CURRENT
    else
        CURRENT = NEXT
}

// we can modify this algorithm to test if whether or
// not CURRENT is a goal state, if optimality isn't
// important, as with k-Queens or SAT
    
```

29

Hill-Climbing for SAT

- How to represent the problem?
 - States, actions, and objective function
- We've seen this before...
 - **States:** binary array that correspond to variable truth assignments (e.g. "1010101010" for 10 variables)
 - **Actions:** toggle a single bit on/off
 - **Objective function:** to minimize the number of unsatisfied clauses

30

Hill-Climbing for k -Queens

- How to represent the problem?
 - States, actions, and objective function
- This is a little tricky...
 - **States:** $k \times k$ chess board with k queens
 - **Actions:** move a single queen to any of its legal positions up, down, or diagonally
 - **Objective function:** minimize the number of conflicts between queens

31

Hill-Climbing for TSP

- How to represent the problem?
 - States, actions, and objective function
- This is even trickier...
 - **States:** an n -city tour (e.g. 1-4-6-2-5-3 is a 6-city tour)
 - **Actions:** swap any two cities in the tour (so the tour is still valid given the problem definition)
 - **Objective function:** minimize the total cost or length of the entire tour

32

Hill-Climbing Issues

- The solution found by HC is totally determined by the initial state
 - How should it be initialized?
 - Should it be fixed or random?
 - Maybe we just want to get started *somewhere* in the search space...
- Can frequently get stuck in local optima or plateaux, without finding the global optimum

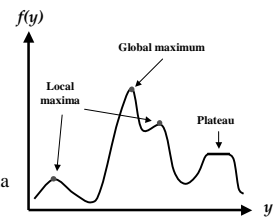
33

Objective Surfaces

The objective surface is a plot of the objective function's "landscape"

The various levels of optimality can be seen on the objective surface

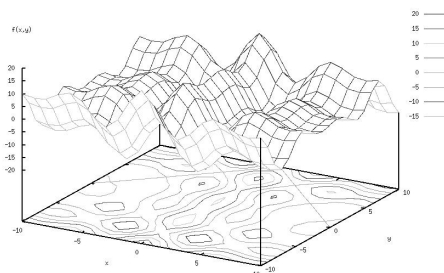
Getting stuck in local optima can be a major problem!



34

Example Objective Surface

$$f(x,y) = -(|x| - 10) \times \cos(x) - y \times \cos(|y| - 10)$$



35

Escaping Local Optima

* *Searching with HC is like scaling Mount Everest in a thick fog with one arm and amnesia*

- Local optima are OK, but sometimes we want to find the *absolute best* solution
- Ch. 5 & 6 of *How to Solve It: Modern Heuristics* have a better discussion of techniques for escaping local optima than *AI: A Modern Approach*

36

Escaping Local Optima

- There are several ways we can try to avoid local optima and find more globally optimal solutions:
 - Random Restarting
 - Simulated Annealing
 - Tabu Search

37

Random Restarting

- * *If at first you don't succeed, try, try again!*
- The idea here is to run the standard HC search algorithm several times, each with different, *randomized* initial states
- Of course, depending on the state space, this can be a difficult task in and of itself... not *all* states that can be generated are *legal* for some problems

38

Random Restarting

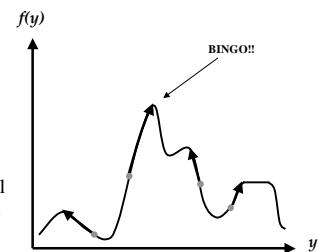
- If the object is to find a *particular* solution (or that reaches a certain goal), we can stop when it is found
- If the aim is to find the *best* solution (optimize), repeat for a fixed number of trials and return the best found

39

Random Restarting

Since HC is a local search strategy, trying multiple initial states allows us to locally explore a wider range of the search space

If we pick lucky initial states, we can find the global optimum!



40

Random Restarting

- As the number of trials increases, the probability of finding a solution (or the global optimum) approaches 1.0
 - This is for the trivial reason that, given enough restarts, we'll ultimately generate the optimum
- But we don't want to keep restarting ad infinitum
 - That would defeat the point of local search!

41

Random Restarting

- It turns out that, if each HC run has a probability p of success, the number of restarts needed is approximately $1/p$
- For example, with 8-Queens, there is a probability of success $p \approx 0.14 \approx 1/7$
- So, on average, we would need only 7 randomly-initialized trails of the basic HC search to find a solution

42

Random Restarting

- Random restart approaches are built in to many state-of-the-art constraint satisfaction algorithms
- They've been shown especially useful in systems geared toward solving hard SAT problems
 - GSAT
 - Davis-Putnam (DPLL with restarts)

43

Simulated Annealing (SA)

- We don't always want to take the best local move, sometimes we might want to:
 - Try taking uphill moves that aren't *the best*
 - Actually go *downhill* to escape local optima
- We can alter HC to allow for these possibilities:
 - Modify *how* successor states are selected
 - Change the *criteria* for accepting a successor

44

Simulated Annealing (SA)

- With standard Hill-Climbing:
 - We explore *all* of the current state's actions/successors
 - Accept the *best* one
- Perhaps we can modify this to account for the other kinds of moves we'd like to make:
 - Choose one action/successor at *random*
 - If it is better, accept it, otherwise accept with some probability p

45

Simulated Annealing (SA)

- These changes allow us to take a variety of new moves, but has problems:
 - Chance of taking a *bad move* is the same at the beginning of the search as at the end
 - The *magnitude* of a move's effect is ignored
- We can replace p with a temperature T which decreases over time
- Since T "cools off" over the course of search, we call this approach simulated annealing

46

Simulated Annealing (SA)

Concepts behind the SA analogy:

Physical system	Optimization problem
Physical state	Feasible solution
Energy	Objective function
Ground state	Goal or optimum
Rapid quenching	Local search
Temperature	Control parameter T
Annealing	Simulated annealing

47

Simulated Annealing (SA)

Let $\Delta E = \text{score}(\text{NEXT}) - \text{score}(\text{CURRENT})$
 $p = e^{\Delta E/T}$ (Boltzman equation)

- $\Delta E \rightarrow -\infty, p \rightarrow 0$
The worse a move is, the probability of taking it decreases exponentially
- Time $\rightarrow \infty, T \rightarrow 0$
As time increases, the temperature decreases, in accordance with a cooling schedule
- $T \rightarrow 0, p \rightarrow 0$
As temperature decreases, the probability of taking a bad move also decreases

48

Simulated Annealing (SA)

```
CURRENT = initialState // initialize the search
for TIME = 1 to ∞ do {
  T = schedule(TIME) // elapsed time effects schedule
  if T = 0 then // T has totally "cooled"
    return CURRENT
  NEXT = random successor of CURRENT
  ΔE = score(NEXT) - score(CURRENT)
  if ΔE > 0 then
    CURRENT = NEXT // take all "good" moves
  else
    CURRENT = NEXT with probability e-(ΔE/T)
}
```

49

Simulated Annealing (SA)

- Can perform downhill and locally sub-optimal moves, unlike HC
- Chance of finding global optimum increased
- SA is fast in practice
 - Only one random neighbor generated per step
 - Only score one successor instead of whole neighborhood
 - Can use more complex heuristics

50

Simulated Annealing (SA)

- According to thermodynamics, to grow a crystal:
 - Start by heating a row of materials in a molten state
 - The *crystal melt* is cooled until it is *frozen in*
 - If the temperature is reduced too quickly, irregularities occur and it does not reach its ground state (e.g. more energy is trapped in the structure)
- By analogy, SA relies on a good cooling schedule, which maps the current *time* to a temperature *T*, to find the optimal solution
 - Usually exponential
 - Can be very difficult to devise

51

Simulated Annealing (SA)

- SA was first used to solve layout problems for VLSI (very large-scale integration) computer architectures in the 1980s
 - Optimally fitting hundreds of thousands of transistors into a single compact microchip
- It is also proven useful for the TSP, and is used in many factory scheduling software systems

52

Tabu Search

- Tabu search is a way to add memory to a local search strategy, and force it to explore new areas of the search space
- We've seen state-based memory before with the closed list, but this memory:
 - Tracks actions taken rather than states expanded
 - Is designed to be a limited (short-term) memory
- Moves that have been seen or taken too recently or too often become *tabu* (or *taboo*)

53

Tabu Search

- We maintain an array *M* which tracks time-stamps of the actions we've taken
 - We store in location *M_i* the most recent time action *i* was taken in the search
- The key parameter of tabu search is the horizon: how long should a certain remain tabu?
 - If we set this too small, we may default to normal HC and stay stuck in local optima
 - If we set it too large, we may run out of legal moves!
 - Usually problem-dependent

54

Tabu Search

```
CURRENT = initialState // initialize search
BEST = CURRENT // retain best solution so far
for TIME = 1 to MAX_TIME do {
    NEXT = best legal successor of CURRENT
    ACTION = action that generated NEXT
    M[ACTION] = tabu info based on horizon & TIME
    CURRENT = NEXT // take next move regardless
    if score(CURRENT) better than score(BEST) then
        BEST = CURRENT
}
return BEST
```

55

Tabu Search

- Instead of an array, memory can also be stored in a queue, or tabu list:
 - As a move is made, place it in the queue
 - When the queue becomes full, the oldest move is removed and becomes legal again
 - The *size* of the queue is the *horizon*

56

Tabu Search

- Since we take the best non-tabu move at each step, we are allowed to take backward steps or just OK moves, as with simulated annealing
- Tabu search can also be faster than standard HC, as it doesn't have to evaluate all action/successors, just those that are *legal*

57

Tabu Search

- Breaking rules
 - Since we retain the best solution so far, we sometimes might want to make tabu moves anyway... if they are better than anything we've previously seen
 - This is called the aspiration criteria, if a tabu solution *aspires* to be better than all previously seen solutions

58

Summary

- Local search methods are more appropriate for solving complete search and optimization problems
 - State spaces can be prohibitively large
 - The goal is different than with partial search strategies
- However, basic hill-climbing can become stuck in local optima rather than finding the best solution

59

Summary

- There are several effective ways of escaping local optima for local searching, which exploit different properties:
 - *Random restarting* tries several times from different parts of the search space
 - *Simulated annealing* allows for a variety of moves by searching stochastically
 - *Tabu search* is deterministic, but incorporates memory to force exploration of the state space

60