Informed search algorithms
Announcements

- Sorry for being sick.
- Teams dividing up the work
  - I have heard rumours that some grad students had to do all the work for the project proposal
  - If you aren’t prepared to make an equal contribution with the rest of your team then you should drop the class
  - I either already know who you are, or I will soon
- Will return project proposals with comments on Wed night. In some cases more detailed versions of those proposals should be turned in a week later. I will let you know who that applies to by a cutoff on points.
Pop Quiz, right now 10 minutes

- What is your name. List the time(s) your team has arranged to meet every week. List the members of your team by first and last name (or email).
- Name the modules that a dialogue system must typically have
- Describe in one sentence the input and the output for each of these modules
- Describe in up to 10 sentences why you were asked to produce a corpus of utterances for your bot, how you did that, and what you will use it for.
- Give a list of three things that might count as features/capabilities that would make your system count as intelligent
- List one or more good performance metrics that your team intends to use to evaluate your BOT as a whole, and say WHY.
Material

- Chapter 4 Section 1 - 3
- Exclude memory-bounded heuristic search
Outline

- Best-first search
- Greedy best-first search
- A* search
- Heuristics
- Local search algorithms
- Hill-climbing search
- Simulated annealing search
- Local beam search
- Genetic algorithms
Review Tree Search

- Difference in search strategies defined by the criteria used to CHOOSE the order of expansion
Best-first search

- Idea: use an evaluation function $f(n)$ for each node
  - estimate of "desirability"

  → Expand most desirable unexpanded node

- Implementation:
  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A* search
Romania with step costs in km
Greedy best-first search

- Evaluation function \( f(n) = h(n) \) (heuristic)
- \( = \) estimate of cost from \( n \) to goal

- e.g., \( h_{SLD}(n) = \) straight-line distance from \( n \) to Bucharest

- Greedy best-first search expands the node that appears to be closest to goal
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Greedy best-first search example
Properties of greedy best-first search

- **Complete?** No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

- **Time?** $O(b^m)$, but a good heuristic can give dramatic improvement

- **Space?** $O(b^m)$ -- keeps all nodes in memory

- **Optimal?** No
A* search

- Idea: avoid expanding paths that are already expensive

- Evaluation function $f(n) = g(n) + h(n)$

- $g(n) = \text{cost so far to reach } n$
- $h(n) = \text{estimated cost from } n \text{ to goal}$
- $f(n) = \text{estimated total cost of path through } n \text{ to goal}$
A* search example
A* search example
A* search example
A* search example
A* search example
A* search example
Admissible heuristics

- A heuristic $h(n)$ is **admissible** if for every node $n$, $h(n) \leq h^*(n)$, where $h^*(n)$ is the **true** cost to reach the goal state from $n$.

- An admissible heuristic **never overestimates** the cost to reach the goal, i.e., it is **optimistic**

- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

- **Theorem**: If $h(n)$ is admissible, A* using **TREE-SEARCH** is optimal
Optimality of A* (proof)

- GO OVER THIS ON YOUR OWN!
- Suppose some suboptimal goal $G_2$ has been generated and is in the fringe. Let $n$ be an unexpanded node in the fringe such that $n$ is on a shortest path to an optimal goal $G$.

\begin{itemize}
  \item $f(G_2) = g(G_2)$ since $h(G_2) = 0$
  \item $g(G_2) > g(G)$ since $G_2$ is suboptimal
  \item $f(G) = g(G)$ since $h(G) = 0$
  \item $f(G_2) > f(G)$ from above
\end{itemize}
Optimality of A* (proof continued)

- Suppose some suboptimal goal \( G_2 \) has been generated and is in the fringe. Let \( n \) be an unexpanded node in the fringe such that \( n \) is on a shortest path to an optimal goal \( G \).

- \( f(G_2) > f(G) \) from above
- \( h(n) \leq h^*(n) \) since \( h \) is admissible
- \( g(n) + h(n) \leq g(n) + h^*(n) \)
- \( f(n) \leq f(G) \)

Hence \( f(G_2) > f(n) \), and A* will never select \( G_2 \) for expansion.
Consistent heuristics

- A heuristic is **consistent** if for every node \( n \), every successor \( n' \) of \( n \) generated by any action \( a \),

\[
h(n) \leq c(n,a,n') + h(n')
\]

- If \( h \) is consistent, we have

\[
f(n') = g(n') + h(n')
= g(n) + c(n,a,n') + h(n')
\geq g(n) + h(n)
= f(n)
\]

- i.e., \( f(n) \) is non-decreasing along any path. (can’t get back costs by going around a “longer” way, effectively prohibits what would seem like negative distances/costs)

- **Theorem**: If \( h(n) \) is consistent, A* using `GRAPH-SEARCH` is optimal
Optimality of A* (go over this in book on your own)

- A* expands nodes in order of increasing $f$ value
- Gradually adds "$f$-contours" of nodes
- Contour $i$ has all nodes with $f=f_i$, where $f_i < f_{i+1}$
Properties of A*

- **Complete?** Yes (unless there are infinitely many nodes with $f \leq f(G)$)
- **Time?** Exponential
- **Space?** Keeps all nodes in memory
- **Optimal?** Yes
Admissible heuristics

E.g., for the 8-puzzle:

- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = total Manhattan distance (i.e., no. of squares from desired location of each tile)

- \( h_1(S) = ? \)
- \( h_2(S) = ? \)
Admissible heuristics

E.g., for the 8-puzzle:

- $h_1(n) = \text{number of misplaced tiles}$
- $h_2(n) = \text{total Manhattan distance}$
  (i.e., no. of squares from desired location of each tile)

- $h_1(S) = 8$
- $h_2(S) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$
Dominance

- If \( h_2(n) \geq h_1(n) \) for all \( n \) (both admissible)
- then \( h_2 \) dominates \( h_1 \)
- \( h_2 \) is better for search

Typical search costs (average number of nodes expanded):

- \( d=12 \)
  - IDS = 3,644,035 nodes
  - \( A^*(h_1) = 227 \) nodes
  - \( A^*(h_2) = 73 \) nodes
- \( d=24 \)
  - IDS = too many nodes
  - \( A^*(h_1) = 39,135 \) nodes
  - \( A^*(h_2) = 1,641 \) nodes
Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.

- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.
Local search algorithms

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution.

- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it
Example: $n$-queens

- Put $n$ queens on an $n \times n$ board with no two queens on the same row, column, or diagonal.
Hill-climbing search

- "Like climbing Everest in thick fog with amnesia"

```
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
```
Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima
Hill-climbing search: 8-queens problem

- $h =$ number of pairs of queens that are attacking each other, either directly or indirectly
- $h = 17$ for the above state
Hill-climbing search: 8-queens problem

- A local minimum with $h = l$
Simulated annealing search

- Idea: escape local maxima by allowing some "bad" moves but **gradually decrease** their 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 prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for i ← 1 to ∞ do
        T ← schedule[i]
        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 $e^{ΔE/T}$
```
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.
Local beam search

- Keep track of \( k \) states rather than just one

- Start with \( k \) randomly generated states

- At each iteration, all the successors of all \( k \) states are generated

- If any one is a goal state, stop; else select the \( k \) best successors from the complete list and repeat.
Ended Here in 1/31. Start here on 2/2
Genetic algorithms

- http://megaswf.com/serve/102223/
- A successor state is generated by combining two parent states
- Start with $k$ randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation
Genetic algorithms (8 queens)

327 42411 + 247 4 8 5 5 2 = 327 4 8 5 5 2
Genetic algorithms (8 queens problem)

- Fitness function: number of non-attacking pairs of queens
- (min = 0, max = $8 \times 7/2 = 28$)
- $24/(24+23+20+11) = 31\%$
- $23/(24+23+20+11) = 29\%$ etc
Genetic algorithms

+ =

[Diagram of chessboard with pieces]
Evolved Virtual Creatures

Examples from work in progress
Adversarial Search & Games
Search versus Games

- Search – no adversary
  - Solution is (heuristic) method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities

- Games – adversary
  - Solution is strategy
    - strategy specifies move for every possible opponent reply.
  - Time limits force an *approximate* solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers, tic-tac-toe, backgammon, bridge
Games as Search

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets reward, loser gets penalty.
  - “Zero sum” means the sum of the reward and the penalty is a constant.

- Formal definition as a search problem:
  - **Initial state**: Set-up specified by the rules, e.g., initial board configuration of chess.
  - **Player(s)**: Defines which player has the move in a state.
  - **Actions(s)**: Returns the set of legal moves in a state.
  - **Result (s,a)**: Transition model defines the result of a move.
  - (2nd ed.: **Successor function**: list of (move, state) pairs specifying legal moves.)
  - **Terminal-Test (s)**: Is the game finished? True if finished, false otherwise.
  - **Utility function(s,p)**: Gives numerical value of terminal state s for player p.
    - E.g., win (+1), lose (-1), and draw (0) in tic-tac-toe.
    - E.g., win (+1), lose (0), and draw (1/2) in chess.

- MAX uses search tree to determine next move.
An optimal procedure: The Min-Max method

Designed to find the optimal strategy for Max and find best move:

1. Generate the whole game tree, down to the leaves.
2. Apply utility (payoff) function to each leaf.
3. Back-up values from leaves through branch nodes:
   - a Max node computes the Max of its child values
   - a Min node computes the Min of its child values
4. At root: choose the move leading to the child of highest value.
Figure 5.2  A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX’s best move is $A_1$, and MIN’s best reply is $A_{11}$. 
Two-Ply Game Tree
Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max

The minimax decision

The diagram shows a two- ply game tree with MAX at the top, MIN at the bottom, and various branches representing different outcomes and decisions.
Pseudocode for Minimax Algorithm

**function** MINIMAX-DECISION(state) **returns** an action
**inputs:** state, current state in game
**return** arg max\(_a\in\text{ACTIONS}(state)\) MIN-VALUE(Result(state, a))

**function** MAX-VALUE(state) **returns** a utility value
  **if** TERMINAL-TEST(state) **then return** UTILITY(state)
  \(v \leftarrow -\infty\)
  **for** \(a\) **in** ACTIONS(state) **do**
    \(v \leftarrow \text{MAX}(v, \text{MIN-VALUE(Result}(state, a)))\)
  **return** \(v\)

**function** MIN-VALUE(state) **returns** a utility value
  **if** TERMINAL-TEST(state) **then return** UTILITY(state)
  \(v \leftarrow +\infty\)
  **for** \(a\) **in** ACTIONS(state) **do**
    \(v \leftarrow \text{MIN}(v, \text{MAX-VALUE(Result}(state, a)))\)
  **return** \(v\)
Properties of minimax

- **Complete?**
  - Yes (if tree is finite).

- **Optimal?**
  - Yes (against an optimal opponent).
  - Can it be beaten by an opponent playing sub-optimally?
    - No. (Why not?)

- **Time complexity?**
  - $O(b^m)$ (IMPRactical FOR MOST GAMES!)

- **Space complexity?**
  - $O(bm)$ (depth-first search, generate all actions at once)
  - $O(m)$ (depth-first search, generate actions one at a time)
Game Tree Size

- Chess
  - $b \approx 35$ (approximate average branching factor)
  - $d \approx 100$ (depth of game tree for “typical” game)
  - $b^d \approx 35^{100} \approx 10^{154}$ nodes!!
  - exact solution completely infeasible

- It is usually impossible to develop the whole search tree for most interesting games
Static (Heuristic) Evaluation Functions

- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent’s score from the player’s.
  - Chess: Value of all white pieces - Value of all black pieces
  - Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
  - If the board evaluation is X for a player, it’s -X for the opponent
    - “Zero-sum game”
Evaluation Functions, cont

For chess, typically \textit{linear} weighted sum of features

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

\[ \text{e.g., } w_1 = 9 \text{ with } f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Alpha-Beta Pruning: Exploiting existence of an Adversary

- If a position is provably bad:
  - It is NO USE expending search time to find out exactly how bad

- If the adversary can force a bad position:
  - It is NO USE expending search time to find out the good positions that the adversary won’t let you achieve anyway

- Bad = not better than we already know we can achieve elsewhere.

- Contrast normal search:
  - ANY node might be a winner.
  - ALL nodes must be considered.
  - (A* avoids this through knowledge, i.e., heuristics)
Alpha-Beta on 2-ply tree we saw earlier

Do DF-search until first leaf

Range of possible values

\([-\infty, +\infty]\)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)

MAX

MIN

[3,3]

[3, +∞]
Alpha-Beta Example (continued)

This node is worse for MAX
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)

MAX

MIN

[3,3]

[3,5] ≥ 3, ≤ 5

[−∞,2] ≤ 2

[−∞,5]

X

X

[3,5]

[3,3]

[3,5]

[12,2]

[8,5]

[3,12]

[8,14]

[2,5]
Alpha-Beta Example (continued)
Alpha-Beta Example (continued)
General alpha-beta pruning: read pp 167-189

- Consider a node $n$ in the tree.
- If player has a better choice at:
  - Parent node of $n$
  - Or any choice point further up
- Then $n$ will never be reached in play.
- Hence, when that much is known about $n$, it can be pruned.
Deterministic games in practice

- **Checkers**: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.

- **Chess**:
  - Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation
  - Uses iterative-deepening alpha-beta search with transpositioning
  - Can explore beyond depth-limit for interesting moves
  - Undisclosed methods for extending some lines of search up to 40 ply.

- **Go**: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
The University of Alberta GAMES Group

Game-playing,
Analytical methods,
Minimax search and
Empirical Studies

Projects | News | What We Do | People | Links | Publications

Announcements

- Weekly GAMES group meetings are from 4-5pm on Thursdays at CSC333. You can check the schedule [here](#).
- The University of Alberta GAMES Group has an opening for a postdoctoral fellow in the area of Artificial Intelligence in Commercial (Video) Games. Check [here](#) for details.

Projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkers</td>
<td>Chinook is the official world checkers champion.</td>
</tr>
<tr>
<td>Poker</td>
<td>Poli is the strongest poker AI in the world.</td>
</tr>
<tr>
<td>Lines of Action</td>
<td>YL &amp; Mova are two of the best LoA programs in the world.</td>
</tr>
<tr>
<td>Hex</td>
<td>Qu examine is one of the best Hex programs in the world.</td>
</tr>
<tr>
<td>Go</td>
<td>The Computer Go group has developed two programs, Explorer and NeuroGo.</td>
</tr>
<tr>
<td>Real-Time Strategy</td>
<td>We are trying to apply AI to real-time strategy games.</td>
</tr>
<tr>
<td>Othello</td>
<td>ExagoBolt defeated the human world Othello champion, 6-0, in 1997. Koyano is another strong program.</td>
</tr>
<tr>
<td>Shogi</td>
<td>Us Shogi has won the world computer Shogi championship many times (currently inactive).</td>
</tr>
<tr>
<td>RoShamBo</td>
<td>Home of the International RoShamBo Programming Competition</td>
</tr>
<tr>
<td>Amazons</td>
<td>These programs, and several theoretical contributions.</td>
</tr>
<tr>
<td>Spades &amp; Hearts</td>
<td>Spades &amp; Hearts are the test beds for the research on multi-player games.</td>
</tr>
<tr>
<td>Post's Correspondence Problem</td>
<td></td>
</tr>
<tr>
<td>Dominor</td>
<td></td>
</tr>
</tbody>
</table>
Summary

- Game playing is best modeled as a search problem.
- Game trees represent alternate computer/opponent moves.
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax chooses moves by assuming that the opponent will always choose the move which is best for them.
- Alpha-Beta can prune large parts of the search tree and allow search to go deeper.
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.