Section

Matthew Tong
Logistics

• Should have HW #1 and Project #1 back
  – See me afterwards if not

• Midterm on May 10
  – Next week’s discussion will mostly be review? Or do you prefer covering logic and maybe an additional session?

• Next project is on games
  – Somewhat of a gap until after logic in terms of fun projects
  – Plus an AI course w/o games would be sad

• Read the discussion board
  – Sometimes important announcements
Project #1

• Grades have two parts, written and performance.
• Written:
  – 10 points writeup style
  – 10 points coding style
  – 10 points testing
  – 5 points graph
  – 15 points approach (Description of algs, correctness of approach, validity of conclusions).
• Performance:
  – 10 Followed directions for running
  – 20 Ran to completion for each, finding solution if there was one
  – 20 Correct results
• Extra:
  - 10 points by original time
  - 3–5 points state checking (depending on implementation)
  - 3 pts IDA*
  - 2 pts per heuristic that's dominates by Manhattan distance (ie worse)
  - 5 pts for better heuristic (no one got this)
  - 2 pts for Iterative/recursive depth first comparison

• Penalties:
  - 1 point to coding for repeated code (no solver abstraction)
• So far, the mean is around 70%. But there'll be some flux and a final report will be coming out once this has stabilized. A few scores w/o reports or who only implemented depth first are affecting the average a lot.
• Interestingly, projects with teams tended to do worse.
• A number of people said things like "ID (or A*) doesn't always find the optimal solution."
  – Of the methods we had you do, only DFS wasn't complete and optimal under these conditions, and knowing this is part of what we expect you to know – so this should have been reported as a bug, not a claim.
  – This is the sort of thing that might be on the midterm.
• Obvious take home message: heuristics help!
• One take home message from the optional section: the importance of pruning (in this case for repeated states)
• These two messages are repeated a lot!
Here's a useful structure for your report to make sure you touch on everything (and make it easier to grade). Use it!

1. Files submitted
   - with brief description of each
2. How to run
   - including unpacking, compiling
3. Approach
   - describe algorithm(s)
     - Shouldn’t HAVE to pour through code
     - Should include what data structures are used
   - problems encountered
   - hurdles overcome
4. Known bugs
   - hopefully empty!
   - But better to report than for me to find.
5. Extra credit
   - can be empty
   - just because you miss it doesn't mean you'll get points,
     but won't get points if not it's NOT listed

6. Sample output

7. Testing
   - convince me (and yourself) it works!
     • Think industry product testing
   - be thorough
     • Bugs in AI are often devious

8. Answers to questions

9. Summary
Minimum remaining values (MRV):
choose the variable with the fewest legal values
Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
choose the variable with the most constraints on remaining variables
• For Sudoku, this corresponds to the number of empty squares in corresponding:
  – Row
  – Column
  – Box
  – Although should probably remove duplicates (some of variables in row/column also in box)

• So when combined with mrv, you typically have where one of those three (or some combo) confines one variable to a small number of values, but that there’s still a lot of uncertainty about those rows, columns, and boxes.
Least constraining value

Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables

Combining these heuristics makes 1000 queens feasible
These three constraints go together
  – “What variable do I pick?”
    • The one with the minumum remaining values
  – “But there’s a tie!”
    • So pick the one with the highest degree
  – “Now how do I pick the value?”
    • Pick the least constraining one

Really *meant* to be used together. But assignment has you also trying individually to get a better feel for them. May or may not help in isolation. (This is science!).
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency

Simplest form of propagation makes each arc consistent

\(X \rightarrow Y\) is consistent iff

for every value \(x\) of \(X\) there is some allowed \(y\)
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment
k-consistency

- A CSP is k-consistent if for any set of k-1 variables and any constraint assignment to those, a consistent value can be assigned to any kth variable
- 1-consistency would just check nodes
- 2-consistency is arc consistency
- 3-consistency (path consistency) - Any pair of variables can always be extended to a third neighboring variable
  - Would catch that WA=red, NSW = red is inconsistent
Strong k-consistency

• Strong k-consistency if 1 … k consistent
• If n variables and doing k-consistency, no search is necessary
  – But exponential!!
  – Middle ground is key
• General point is general CSP. But this is where domain knowledge can figure in.
  – In Sudoku, X-wing configuration is a kind of 3-consistency
<table>
<thead>
<tr>
<th>23</th>
<th>1</th>
<th>23</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Iterative algorithms for CSPs

Hill-climbing, simulated annealing typically work with
“complete” states, i.e., all variables assigned

To apply to CSPs:
- allow states with unsatisfied constraints
  - operators reassign variable values

Variable selection: randomly select any conflicted variable

Value selection by min-conflicts heuristic:
- choose value that violates the fewest constraints
  - i.e., hillclimb with $h(n) =$ total number of violated constraints
Other important techniques

• Exploit structure
  – If tree, can get linear
  – If tree-like, can decompose and get linear

• More intelligent backtracking
  – By default “chronological backtracking”
  – But really want to back track to point that caused error
Summary

CSPs are a special kind of problem:
   states defined by values of a fixed set of variables
   goal test defined by constraints on variable values

Backtracking = depth-first search with one variable assigned per node

Variable ordering and value selection heuristics help significantly

Forward checking prevents assignments that guarantee later failure

Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

The CSP representation allows analysis of problem structure

Tree-structured CSPs can be solved in linear time

Iterative min-conflicts is usually effective in practice
Game Playing
Games vs. search problems

“Unpredictable” opponent ⇒ solution is a strategy specifying a move for every possible opponent reply

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)
<table>
<thead>
<tr>
<th>Types of games</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>perfect information</strong></td>
</tr>
<tr>
<td>chess, checkers, go, othello</td>
</tr>
<tr>
<td>imperfect information</td>
</tr>
</tbody>
</table>
Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
     = best achievable payoff against best play

E.g., 2-ply game:

```
MAX

A_1

A_2

A_3

MIN

A_{11}  A_{12}  A_{13}

A_{21}  A_{22}  A_{23}

A_{31}  A_{32}  A_{33}

3       12       8

2       4        6

14      5        2
```
Properties of minimax

**Complete??** Yes, if tree is finite (chess has specific rules for this)

**Optimal??** Yes, against an optimal opponent. Otherwise??

**Time complexity??** $O(b^m)$

**Space complexity??** $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games

$\Rightarrow$ exact solution completely infeasible

But do we need to explore every path?
$\alpha-\beta$ pruning example
\( \alpha-\beta \) pruning example
\( \alpha - \beta \) pruning example
α–β pruning example
$\alpha-\beta$ pruning example

MAX

MIN

3 12 8 2

\( \leq 2 \)

\( \times \times 2 \)
Why is it called $\alpha-\beta$?

$\alpha$ is the best value (to MAX) found so far off the current path. If $V$ is worse than $\alpha$, MAX will avoid it $\Rightarrow$ prune that branch. Define $\beta$ similarly for MIN.
Evaluation functions

Black to move
White slightly better

White to move
Black winning

For chess, typically linear weighted sum of features

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

e.g., $w_1 = 9$ with
\[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.} \]
Horizon effect

• So you run evaluation function on leaves
• But what if something happens just “beyond the horizon” - such as next step
• One solution: Only stop at “quiescent positions”
  – Such as where no piece captured on next move
• Can also have long sequences where something is inevitable (say the taking of a queen) but one side takes actions to stall the inevitable. Hard to eliminate, but deeper searches make rare.
Deterministic games in practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of $443,748,401,247$ positions.


Othello: human champions refuse to compete against computers, who are too good.

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.
Nondeterministic games in general

In nondeterministic games, chance introduced by dice, card-shuffling

Simplified example with coin-flipping:

```
      MAX
     /   \
   CHANCE  MIN
  /   \     /   \
3  0.5  0.5  0.5  0.5
 / \ / \   / \ / \   / \ / \ 
2  4 7 2  0 4 6 0 5 -2
```

Games of imperfect information

E.g., card games, where opponent’s initial cards are unknown

Typically we can calculate a probability for each possible deal

Seems just like having one big dice roll at the beginning of the game*

Idea: compute the minimax value of each action in each deal,
      then choose the action with highest expected value over all deals*

Special case: if an action is optimal for all deals, it’s optimal.*

GIB, current best bridge program, approximates this idea by
   1) generating 100 deals consistent with bidding information
   2) picking the action that wins most tricks on average
Commonsense example

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you’ll find a mound of jewels;
    take the right fork and you’ll be run over by a bus.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    take the left fork and you’ll be run over by a bus;
    take the right fork and you’ll find a mound of jewels.

Road A leads to a small heap of gold pieces
Road B leads to a fork:
    guess correctly and you’ll find a mound of jewels;
    guess incorrectly and you’ll be run over by a bus.
* Intuition that the value of an action is the average of its values in all actual states is **WRONG**

With partial observability, value of an action depends on the **information state** or **belief state** the agent is in.

Can generate and search a tree of information states.

Leads to rational behaviors such as
- Acting to obtain information
- Signalling to one’s partner
- Acting randomly to minimize information disclosure
Summary

Games are fun to work on! (and dangerous)

They illustrate several important points about AI

◊ perfection is unattainable ⇒ must approximate
◊ good idea to think about what to think about
◊ uncertainty constrains the assignment of values to states
◊ optimal decisions depend on information state, not real state

Games are to AI as grand prix racing is to automobile design