Search, Backtracking, Heuristics

- **How do you find a needle in a haystack?**
  - How does a computer play chess?
  - Why would you write that program?

- **How does Mapquest/Googlemap find routes from one place to another?**
  - Shortest path algorithms
  - Longest path algorithms

- **Why are these videos relevant? Copyright issues?**
  - [http://www.youtube.com/watch?v=YKSIaeQHV94](http://www.youtube.com/watch?v=YKSIaeQHV94)
Exhaustive Search/Heuristics

- We use binary search trees to organize data, in searching we don’t need to examine all the data to find what we’re looking for
  - Where is the smallest item in a search tree? Largest?
  - How are trees balanced?

- What do we do when the search space is huge?
  - How many chess boards are there?
  - How many routes are there between my house and yours?

- Exhaustive search: look at everything!
Classic problem: N queens

- Can queens be placed on a chess board so that no queens attack each other?
  - Easily place two queens
  - What about 8 queens?
- Make the board \( N \times N \), this is the N queens problem
  - Place one queen/column
  - Horiz/Vert/Diag attacks
- Backtracking
  - Tentative placement
  - Recurse, if ok done!
  - If fail, undo tentative, retry
- wikipedia-n-queens
Backtracking idea with N queens

- **Try to place a queen in each column in turn**
  - Try first row in column $C$, if ok, move onto next column
  - If solved, great, otherwise try next row in column $C$, place queen, move onto the next column
    - Must unplace the placed queen to keep going

- **What happens when we start in a column, where to start?**
  - If we fail, move back to previous column (which remembers where it is/failed)
  - When starting in a column anew, start at beginning
    - When backing up, try next location, not beginning

- **Backtracking in general, record an attempt go forward**
  - If going forward fails, undo the record and backup
N queens backtracking: Queens.java

public boolean solve(int col) {
    if (col == mySize) return true;
    // try each row until all are tried
    for (int r = 0; r < mySize; r++) {
        if (myBoard.safeToPlace(r, col)) {
            myBoard.setQueen(r, col, true);
            if (solve(col + 1)) {
                return true;
            }
            myBoard.setQueen(r, col, false);
        }
    }
    return false;
}
Basic ideas in backtracking search

- We need to be able to enumerate all possible choices/moves
  - We try these choices in order, committing to a choice
  - If the choice doesn’t pan out we must undo the choice
    - This is the backtracking step, choices must be undoable

- Process is inherently recursive, so we need to know when the search finishes
  - When all columns tried in N queens
  - When we have found the exit in a maze
  - When every possible moved tried in Tic-tac-toe or chess?
    - Is there a difference between these games?

- Summary: enumerate choices, try a choice, undo a choice, this is brute force search: try everything
Pruning vs. Exhaustive Search

- If we consider every possible placement of 4 queens on a 4x4 board, how many are there? (N queens)
  - $4 \times 4 \times 4 \times 4$ if we don’t pay attention to any attacks
  - $4 \times 3 \times 2 \times 1$ if we avoid attacks in same row

- What about if we avoid diagonal attacks?
  - Pruning search space makes more search possible, still could be lots of searching to do!

- Estimate how long to calculate # solutions to the N-queens problem with our Java code....
Queens Details

- **How do we know when it’s safe to place a queen?**
  - No queen in same row, or diagonal
  - For each column, store the row that a queen is in
  - See QBoard.java for details

- **For GUI version, we use a decorator**
  - The QBoardGUI is an IQueenState class and it has an IQueenState object in it
  - Appears as an IQueenState to client, but uses an existing one to help do its work
  - One of many object oriented design patterns, seen in Huff in the BitInputStream class
Computer v. Human in Games

- Computers can explore a large search space of moves quickly
  - How many moves possible in chess, for example?

- Computers cannot explore every move (why) so must use heuristics
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move

- What do humans do well in these games? What about computers?
  - What about at Duke?
Games at Duke

- **Alan Biermann**
  - Natural language processing
  - Compsci 1: Greate Ideas
  - Duchess, checkers, chess

- **Tom Truscott**
  - Duke undergraduate working with/for Biermann
  - Usenet: online community

- **Second EFF Pioneer Award (with Vint Cerf!)**
Heuristics

- **A heuristic is a rule of thumb, doesn’t always work, isn’t guaranteed to work, but useful in many/most cases**
  - Search problems that are “big” often can be approximated or solved with the right heuristics

- **What heuristic is good for Sudoku?**
  - Is there always a no-reasoning move, e.g., 5 goes here?
  - What about “if I put a 5 here, then...”
  - Do something else?

- **What other optimizations/improvements can we make?**
  - For chess, checkers: good heuristics, good data structures
Boggle Program
Boggle Search for Word

• Starting at board location (row, col) to find a string s
  - We want to keep track of where we are in the string
  - We want to keep track of what board locations we’ve used

• How do we know when we’re done?
  - Base case of recursive, backtracking call
  - Where we are in the string?

• How do we keep track of used locations?
  - Store in array list: tentatively use current one, recurse
  - If we don’t succeed, take off the last one stored!
Backtracking, minimax, game search

- We’ll use tic-tac-toe to illustrate the idea, but it’s a silly game to show the power of the method
  - What games might be better? Problems?

- Minimax idea: two players, one maximizes score, the other minimizes score, search complete/partial game tree for best possible move
  - In tic-tac-toe we can search until the end-of-the game, but this isn’t possible in general, why not?
  - Use static board evaluation functions instead of searching all the way until the game ends

- Minimax leads to alpha-beta search, then to other rules and heuristics
Minimax, see TicTac.java

- Players alternate, one might be computer, one human (or two computer players)

- Simple rules: win scores +10, loss scores -10, tie is zero
  - X maximizes, O minimizes

- Assume opponent plays smart
  - What happens otherwise?

- As game tree is explored is there redundant search?
  - What can we do about this?