Day 5: Problem Spaces

Strong Methods

- Strong methods are based on domain knowledge, and are often much faster
- Law of Representation: concepts are mapped into a symbolic form (ie. bits), then the representations are manipulated, then the results are translated back.
- Lots of problems (ex. puzzles) fall into the category of searching a problem space for valid solutions.
    - Ex: Tower of Hanoi, Water Jug Problem, Coin Puzzles (what 5 coins add up to 42c, etc.)
    - Humans use heuristics (ex. trying quarters first, etc.)
    - N Queens problem: place N queens on an NxN chessboard such that none of them could be attacked by any other
- How many configurations are there?
    - Water Jug: 20, Hanoi: n^3, Coin Puzzle (polynomial), etc.
    - Often can't store every possible combination
- The Problem Space Approach:
    - Formalize a task as:
        - a set of discrete states
        - a set of legal operations which transition between the states
        - Initial and Goal States
        - Solving a problem means finding a legal path from the initial to goal states.
    - This is representation & search, as is much of AI
- A graph is a data structure composed of a set of vertices (V) and edges/connections (E).

    - A problem space is a graph too
- The idea is that as the graphs get very big finding the solution through them becomes very complicated
- Trade-offs of Representation and Ruels
    - Compact representation means more CPU work, etc.
- Fun Math Trick: I K is a power of two, then $2^k = 100000 \ \& \ 2^k - 1 = 011111$

Weak Methods

- General purpose methods with no domain knowledge (except sometimes heuristically)
- Weak methods often arise from how humans solve problems
- Examples:
    - Brute force/enumeration:
        - Enumerate every possible solution and check it (like password cracking)
        - This actually works sometimes, although it's frowned upon
    - Generate & Test:
        - Generate a possible solution and test it.
        - Often gets you an answer faster than brute force enmueration.
    - Random:
        - `while currentState != goalState: pick any rule that applies and execute it`
        - Depending on the problem, this sometimes works okayish. Often it gets stuck for many iterations then has a breakthrough.
        - Some problems (like coins) are dissipative, which means you can get to a dead end.
    - Hill Climbing