Machine Planning with WaveFront

Daniel Perano

GOAP: Goal Oriented Action Planning

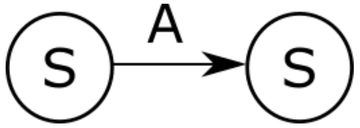
- Starting State
- Goal State
- Actions we can take
- We want the “best” way to attain the goal
- GOAP is domain independent (general)

Example: Pathfinding

- Start Position
- Goal Position
- 1 Action - Move
- A* is specific, but GOAP is generalized to all problems that can be expressed as states and actions

Background: Graph Search Algorithms

- Planning is often expressed as a search over a state graph:



- Graph search is a well studied area with many successful algorithms - A*, Dijkstra, DFS, BFS, Prim, Beam search

- Problem #1: GOAP state spaces are fully generalized

- Problem #2: General state spaces can have a *huge* number of nodes: $O(n!) \rightarrow \infty$

- Problem #3: Graph searches often require specific distance-based heuristics

Background: Completeness & Optimality

- Complete: *Guaranteed* to find a solution if one exists
- Optimal: *Guaranteed* to find the optimal solution
- Problem: Completeness & Optimality force a O(n!) full traversal on a general graph!
- Solution: Approximate Completeness & Optimality - if a solution exists, we will *probably* find it and it will *probably* be optimal or near-optimal

Formulating a GOAP-Specific Search: We want…

- Speed: Fast solve times
- Scalable: Adding actions doesn’t have catastrophic consequences
- Parallel: Find multiple solutions at once
- (Approximately) Complete & (Approximately) Optimal
- Extra Credit! - Tunable parameters to adapt to use cases

WaveFront is…

- Fast
- Scalable
- Parallel
- Conditionally Complete & Conditionally Optimal
- Tunable

WaveFront - Overview

1. Launch the wave front
2. Propagate each wave until goal is reached or wave is terminated:
    1. Simulate all remaining actions
    2. Sort actions by outcome
    3. Choose the best next action
3. Continue until all waves find a solution or are terminated

WaveFront - Launch the Wave Front

- A wave is a potential solution - some will fail and be discarded, some may succeed in reaching the goal state
- The “wave front” is the collection of all waves
- Every wave is created at the first solving step
- How many waves get created depends on how many solutions are desired
- Waves may get stuck and be terminated without reaching a solution

WaveFront - Propagate Waves

1. Simulate all actions that can possibly come after the current wave state
    - If stuck, terminate or backtrack
2. Sort actions in order from best to worst
3. Pick the best action, and set the wave’s current state to this action’s outcome
4. Repeat until the goal is found or the wave gets stuck

WaveFront - Dealing with Stuck Waves

- Waves are stuck when they reach a state where no actions are possible
- Two ways to deal with a stuck wave:
    1. Terminate immediately (fastest)
    2. Backtrack
        - Unwind previous action, and then proceed forwards with the next-best action

WaveFront - Backtracking

- Backtracking usually doesn't end well

    1. It almost always ends in failure. Most of the time, repeatedly choosing the best action will always reach the solution (for some models, this *may* always hold and backtracking can be proven unnecessary), so if you get stuck you'll probably never find a solution.

    2. It's *slow*. If you keep on backtracking without limit you'll eventually traverse the entire state space reachable by the backtracking wave - O(n!)

    3. For completeness' sake, WaveFront supports backtracking for the rare cases where it may be useful. (*Conditional Completeness*)

WaveFront - Analysis

1. Asymptotic dependencies:
    - Number of actions
    - Length (number of actions used) of longest solution
2. “Wall clock” dependencies
    - Cost of simulating actions
    - Cloning states

WaveFront - Analysis

1. Asymptotic performance (time & memory):
   - O(m * n log n), m = length of longest solution, n = number of actions
   - Asymptotic behavior with respect to actions is bounded by the sort algorithm
   - Typical sort algorithms are O(n log n)
   - If m is a known constant, WaveFront's complexity is O(n log n)!
2. Wall clock performance:
   - Action simulation is usually fast OO or functional code
   - Cloning states can be expensive if the implementation language doesn't have well-optimized memory and GC operations - this is a good place to look for opportunities to optimize

WaveFront - Example Performance (no backtracking)

- WaveFront's reference implementation is Python 3 (running on CPython)
- ~6 seconds to solve a 2-step problem with 10k actions
  - ~97% of runtime spent in choosing next action to take
  - ~55% state analysis/cloning / ~32% simulation
  - WaveFront spends almost all its time processing the state model - choosing the next step is *almost* free!
- 0.03s (30 ms) to solve a 2-step problem with 15 actions

WaveFront - Tuning

- Depth Limit
  - Stops the solver from searching for a solution longer than this limit
- Desired Solution Count
  - How many solutions to search for (how many waves are initially launched)
- Maximum Generations
  - How many times a wave can be backtracked before being discarded