Reinforcement Path Finding

By

ANONYMOUS
CPSC 589
Dr. Chang-Hyun Jo

Outline

- Problem of Path Finding
- Path Finding Methods
  - Dijkstra's Algorithm
  - Best First Search
  - A* Algorithm
    - Incremental A*
    - Hierarchical A*
  - D*
- Case Study: Intelligent Route Finding
- Conclusion

Problem of Path Finding

- Problem of Path Finding:
  - Common problem in many fields
    - E.g. robotics, games, web routing
  - Want to find the shortest, most optimal path
  - Desire efficient algorithm to reduce computation needed and computation time.
  - Current methods of path finding use heuristics to find the most optimal path
    - Wastes CPU cycles
    - Proposed solution: Combine reinforcement learning methods and current path finding heuristics

Path Finding Methods

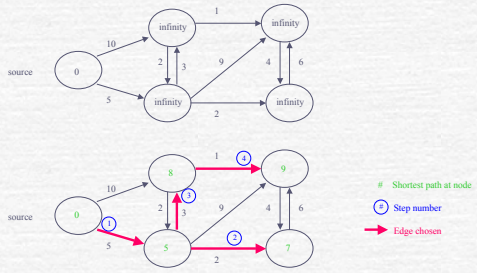
- 2 ways path finding algorithms have been implemented:
  - Path Finding:
    - Find entire path before traversing
    - Advantage:
      - Typically better paths found that Step Taking Algorithms
  - Step Taking:
    - Find path on a step by step basis
    - Advantage:
      - No CPU spikes caused by not doing entire calculation at one time
    - Disadvantage:
      - Agent may choose wrong paths and get stuck

Dijkstra's Algorithm

- Simplest path finding algorithm:
  - Disadvantage:
    - Computation longer than more efficient algorithms
  - Advantage:
    - Find a shortest path from a single source node to all nodes
- Represents a graph using the following notation:
  - G = (V, E)

where G = entire graph
V = set of vertices
E = set of edges

Dijkstra's Algorithm



Best First Search

- Part of a more general category of search strategies (Hill Climbing Strategy)
  - Current state is used to search and evaluate its children
  - Best child is selected and used for further expansion/search
    - Parents are disregarded
    - Siblings are disregarded
  - Advantage:
    - Good for when search space is large
    - Prunes search space considerably if a lot of siblings
  - Disadvantage:
    - may often result in failing paths

Best First Search

- Best First Search:
  - Attempts to find the shortest path by considering the estimates of the best partial solution next
  - Solves problem of Hill Climbing strategies by using back-tracking
  - Typically implemented using a priority queue:
    - Ordered based on a heuristic value assigned to each node
    - Keeps 2 lists: Open and Closed List
      - Open List: Keeps nodes that are being examined
      - Closed List: Keeps nodes that have already been examined
    - Closed List allows for back-tracking

A* Algorithm

- Currently the most used algorithm for path finding
- Preferred over Dijkstra's and Best First Search because it is faster and more efficient
- Uses a heuristic to estimate the optimal path
  - Heuristic consists of:
    - Distance from starting point to an intermediary node
    - Estimation of the distance from the intermediary node to the goal node
  - Represented by the following equation:

$$F(p) = G(p) + H(p)$$

A* Algorithm

- F(p) = heuristic value assigned to a node
- G(p) = distance from the starting node to an intermediary node
- H(p) = estimation from intermediary node to the goal node
- Advantage:
  - Tradeoff can be made in calculating the H(p) value
    - Overestimate:
      - Results in faster computation time
      - Results in less accurate optimal path
    - Underestimate:
      - Results in more accurate optimal path
      - Results in slower computation time

A* Algorithm

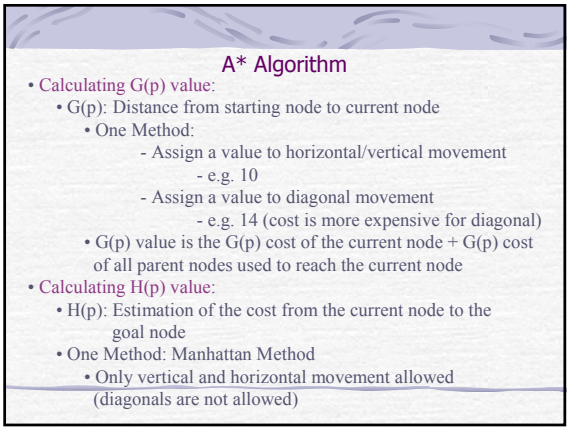
- Steps:
  1. Begin at Starting State (initially on OPEN list).
  2. Add all traversable neighboring nodes to OPEN list.
     - Mark that the parent node of these nodes is the Starting State.
  3. Add Starting Node to the CLOSED list.
     - Choose neighboring node with the lowest F(p) to be next current node.
  4. For the current node, find the F(p) values of all traversable neighboring nodes that aren't already on the CLOSED list.
     - Add these neighboring nodes to the OPEN list.

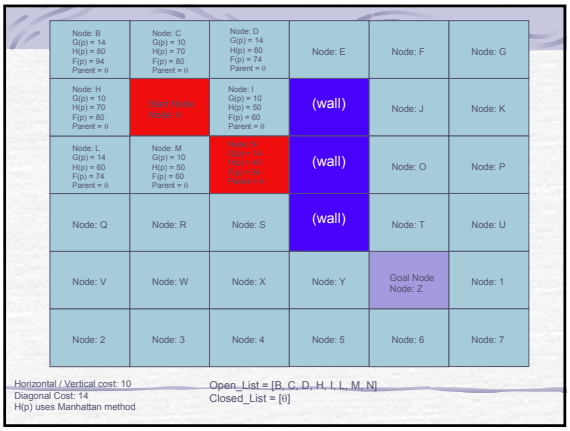
A* Algorithm

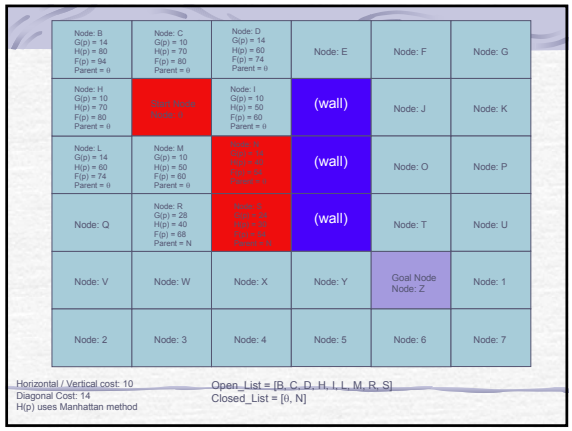
- Steps (cont.):
  5. Add current node to the CLOSED list, removing from the OPEN list.
     - Choose the node with the lowest F(p) value in the OPEN list to be the next current node.
     - Repeat steps 4 and 5 until the goal state is reached.
- When adding a node to the OPEN list:
  - If it already exists in the OPEN list, G(p) value must be recalculated
  - Check if F(p) value is less by reaching node through current node
    - If F(p) value is less, update parent pointer and F(p) values

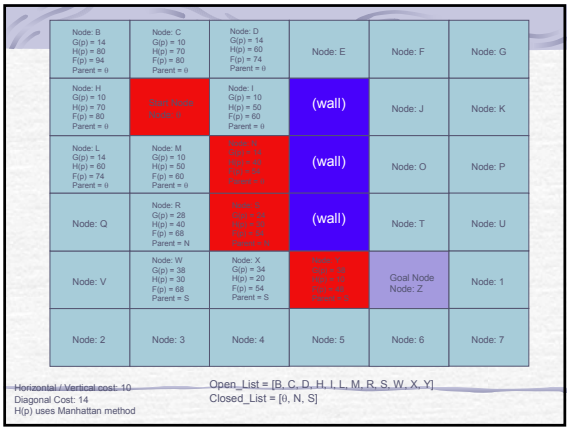
A* Algorithm

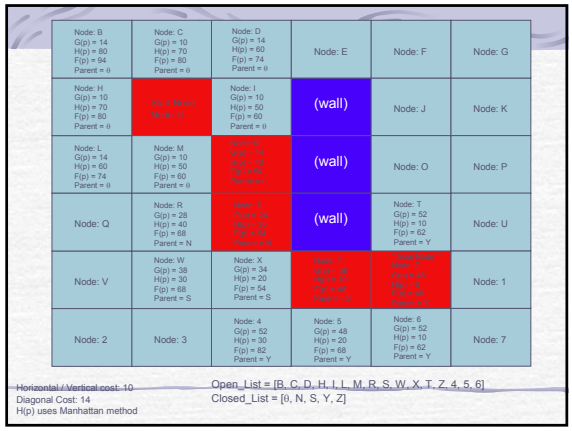
- Calculating G(p) value:
  - G(p): Distance from starting node to current node
    - One Method:
      - Assign a value to horizontal/vertical movement
        - e.g. 10
      - Assign a value to diagonal movement
        - e.g. 14 (cost is more expensive for diagonal)
    - G(p) value is the G(p) cost of the current node + G(p) cost of all parent nodes used to reach the current node
- Calculating H(p) value:
  - H(p): Estimation of the cost from the current node to the goal node
  - One Method: Manhattan Method
    - Only vertical and horizontal movement allowed (diagonals are not allowed)











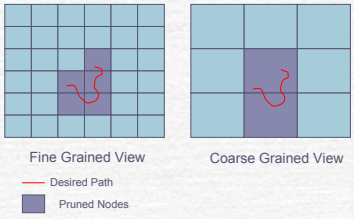
Incremental A* Algorithm

- Attempts to fix some deficiencies of the standard A* algorithm
- Standard A* algorithm requires computation every time a path needs to be found
  - Even if the path had been traversed previously
- Incremental A* uses knowledge from previous paths found to calculate new paths
- First run results in a computation time equal to the original A*
  - Subsequent runs are typically sped up because previous knowledge will exist for the paths

Hierarchical A* Algorithm

- Attempts to speed up standard A* algorithm
- Breaks up search space into varying degrees of levels
  - Start at a broad view
    - Result: Pruning of the search space
  - Go into more detailed view only when necessary and after the search space has been pruned
- One method:
  - Vary the size of the nodes
    - Coarse grained view: Larger Nodes
    - Finer grained view: Smaller Nodes
    - Can have multiple levels

Hierarchical A* Algorithm



Problem with Heuristic Based Approach

- Entire search space must be known for above heuristic based algorithms to work
- Not always possible to know the entire search space
- D* algorithm developed by Anthony Stentz
  - Modeled after the A* algorithm
  - Modified to handle dynamic environments or unknown environments

D* Algorithm

- Uses RAISE and LOWER states to handle a dynamic environment
  - RAISE state: propagates information about cost increases when they are found (e.g. obstacles)
  - LOWER state: propagates information about cost decreases when they are found
- In comparison with the optimal algorithm found, D* resulted in a path that was 2 times that of the optimal path
  - Attributed to not having complete knowledge of the environment
  - Not a equal comparison

D* Algorithm

- Compared D* with Optimal Replanner:
  - Optimal Replanner:
    1. Computes optimal path from start state to goal node
    2. When obstacle is reached, re-computes the optimal path
- Results:

Algorithm	1,000	10,000	100,000	1,000,000
Replanner	427 msec	14.45 sec	10.86 min	50.82 min
D*	261 msec	1.69 sec	10.93 sec	16.83 sec
Speed-Up	1.67	10.14	56.30	229.30

Case Study: Intelligent Route Finding

- All of the algorithms above are wasteful in computation (except (Incremental A*)
  - Reason: Every time a path needs to be found, agent must calculate the path from scratch
- Bing Liu devised an intelligent route finding system
  - Constraints of the problem:
    - System hardware in vehicle has small amount of memory
    - Routes should reflect driver's preferences
- Designed a unique "Disk and Memory" based route finder

Case Study: Intelligent Route Finding

- Disk and Memory Design:
  - stores MJN's (Major Junction Network's) in memory
    - e.g. Freeways, major highways, etc.
    - equivalent to coarse grained view
  - stores DMR's (Detailed Major Road's) on disk
    - equivalent to finer grained view
  - Calculates major path using the MJN
  - Loads in DMR's from disk only when they're necessary
- Based on user feedback, roads are assigned values
  - If a driver expresses dissatisfaction with a road, the system makes the road's cost higher
  - If a driver does not express dissatisfaction, road costs are reduced
  - As a result, when the system picks roads, it chooses lower cost roads

Conclusion

- All previous methods used for path finding have flaws:
  - The majority of heuristic based algorithms assume that complete knowledge of the entire environment is known prior to formulating an optimal path
  - Algorithms that can handle dynamic environments are computation inefficient
  - Intelligent Route Finding case study assumes that the entire street map is known prior to calculating paths

Project Proposal

- Combine the methods of Machine Learning and apply these methods to solve the problem of path finding:
  - Machine learning advantages:
    1. Allows previous traversals and paths to be used to calculate new paths
    2. Allows the agent to actually learn its environment
    3. Computation time should decrease as learning occurs
    4. Allows the agent to determine optimal paths based using a system of reinforcement learning for the agent
  - Currently no heuristic based approaches do this

Error: Need to give a specific research topic proposed!

References

[Ito et al. 2001] Ito, Akira, and Mitsuru Kanabuchi. 2001. Speeding Up Multiagent Reinforcement Learning by Coarse-Graining of Perception: The Hunter Game, Electronics and Communications in Japan, 84 (12), 37-45.

[Koenig 2001] Koenig, Sven. 2001. Glossary of Fast Replanning and Greedy On-Line Planning, http://www.cc.gatech.edu/fac/Sven.Koenig/greedyonline/glossary.html#LifelongPlanningA*.

[Lester 2003] Lester, Patrick. 2003, A* Pathfinding for Beginners, http://www.policyalmanac.org/games/aStarTutorial.htm.

[Linden 1992] Linden, Alexander. 1992. On Discontinuous Q-Functions in Reinforcement Learning, 16th German Conference on Artificial Intelligence, Bonn, Germany (GWAI 92), (August), 199-209.

[Liu 1996] Liu, Bing. 1996. Intelligent Route Finding: Combining Knowledge, Cases, and An Efficient Search Algorithm, 12th European Conference on Artificial Intelligence, Budapest, Hungary, (August), 380-384.

[Luger 2002] Luger, George. 2002. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE.

[Morris 1998] Morris, John. 1998. Data Structures and Algorithms: Dijkstra's Algorithm, http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/dijkstra.html.

[Patel 2001] Patel, Amit. 2001. Amit's A* Pages, http://theory.stanford.edu/~amitp/GameProgramming/.

[Pinter 2001] Pinter, Marco. 2001. Toward More Realistic Pathfinding, http://www.gamasutra.com/features/20010314/pinter_01.htm.

References

[Seri and Tadepalli 2002] Seri, Sandeep and Tadepalli, Prasad. 2002. Model-based Hierarchical Average-reward Reinforcement Learning, International Conference on Machine Learning, Sydney, Australia, (July), 562-569.

[Stentz 1994] Stentz, Anthony. 1994. Optimal and Efficient Planning for Partially-Known Environments, Proceedings IEEE International Conference on Robotics and Automation, San Diego, California, (May), 3310-3317.

[Stentz 2002] Stentz, Anthony. 2002. CD*: A Real-time Resolution Optimal Re-planner for Globally Constrained Problems, Proceedings of the National Conference on Artificial Intelligence (AAAI-02), Edmonton, Alberta, (July), 605-611.

[Tanimoto 1995] Tanimoto, Steven. 1995. The Elements of Artificial Intelligence Using Common Lisp, 2nd Edition. Computer Science Press, New York, New York.