Motion Planning

Graph Based Methods

Jana Kosecka
Department of Computer Science

- Discrete planning, graph search, shortest path, A* methods
- Road map methods
- Configuration space

Slides thanks to http://cs.cmu.edu/~motionplanning, Jyh-Ming Lien

State space

- Set of all possible states is represented as graph
- Nodes states, links possible transitions between the states
- We will consider setting now where
  - State space $\mathcal{X}$ is finite or countable infinite

  - Transition function $x' = f(x, u)$
  - Set of action in each state $U(x)$
  - Initial and goal state

- Feasible planning – generate a set of actions, if the solution exists it must be found in the finite time
- Search must be systematic, explore unexplored states, keep track what has been visited
- Strategy: used graph search algorithms
Motion Planning Problem

- Task planning – doing the dishes
- Motion planning problem how to go from pose one to pose two
- Example PACMAC

Planning on a grid

- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start - green
Planning on a grid

- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start – green
- Graph structure
- Set of nodes $V$
- Set of edges $E$
- $G = (V, E)$

Planning on a graph

- $G = (V, E)$
Planning on a grid

- Robot can move between adjacent cells
- Obstacles - black
- Goal – red
- Start – green
- Graph structure
- Set of nodes $V$
- Set of edges $E$
- $G = (V, E)$
- Here unit distance Between the nodes

Find a path that minimizes cost
- Minimizes distance
- Shortest path
Grassfire Algorithm

- Mark goal with 0
- Mark all neighbours +1
- Until you reach start node

- Create an empty list
  add - goal to the list

While list not empty
Remove the first node
if n.dist = infinity
  n.dist = n.dist +1
add n to the back of the list
- To move to the goal just
  move to the node with smallest
distance values

Another example
Terminates
- If the start node has not been found, algorithm terminates.
Computational Complexity

- Grassfire algorithm
- Complexity linear in the number of nodes

- In 2D grid
  - $100 \times 100 = 10^4$
  - In 3D $100 \times 100 \times 100 = 10^6$
  - In 6D $10^{12}$

- Problem –
  - Exponential complexity in terms of number of degrees of freedom

Dijkstra's Shortest Path Algorithm

Single shortest path – single destination $t$ (single source)
Given pair of vertices – what is the shortest path from $u$ to $v$
Positive weights
Example:
Dijkstra's Shortest Path Algorithm

• Find shortest path from s to t.

Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }

delmin

distance label

Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }

decrease key

distance label
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 6, 7 \}$

$PQ = \{ 3, 4, 5, t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]

\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ PQ = \{ t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
Optimal Substructure

Cost of the path – sum of the weight of all edges

Observation 1: Shortest path has an optimal substructure
Problem (we will see greedy and dynamic programming techniques)

If \( p \) is the shortest path from \( v_1 \) to \( v_k \), \( v_1, v_2, ... v_i, ... v_j, ... v_k \). The for any \( i, j \) the shortest path from \( v_i \) to \( v_j \) is contained in it.

Proof: if some subpath is not a shortest path then we can substitute it and obtain shorter path then the original (contradiction)

Notion of the shortest path is well defined only when there are
No negative weight cycles – we will just consider positive weights

Relaxation

Idea: originally assign the path some upper bound and then keep on decreasing it as until you reach the cost of shortest path, \( d[u] \) – attribute keeps upper bound on the cost of shortest path

Dijkstra's Algorithm (G,w,s) \%(with adjacency lists)\nInitialize-Single-Source(G,s) \% set cost to go to infinity – except \( S := 0 \) \% S start node \( Q := V[G] \) \% priority queue put all vertices there

while \( Q = 0 \) do
  \( u := \text{extract}_\text{min}(Q) \)
  \( S := S U \{u\} \)
  for each vertex \( v \) in \( \text{Adj}[u] \) do
    if \( v \) in \( Q \) and \( d[u] + w(u,v) < d[v] \) then
      \( d[v] := d[u] + w(u,v) \)
  End

Example blackboard
Running time analysis

The values of the array are kept in priority queue (min-heap)
- Updating the heap structure takes at most $O(lg V)$ and there are $|V|$ of such operations. Building the heap takes $O(V)$.
- Extract min, we need to fix heap structure $|V|$ times, $V \ lg \ V$
- Also relaxation needs to Decrease_key for elements in the heap and still fix the heap structure (that can be done in $E \ lg \ V$)
- The total running time:
  $O \left((V+E) \ lg \ V\right)$ for sparse graphs $O \left(E \ lg \ V\right) \quad E \sim V$ sparse graph
  $O \left(V \ lg \ V + E\right)$ with Fibonacci heap decrease key amortized cost is $O(1)$

If the priority queue is maintained as linear array then
Extract_min takes $O(V)$ and there are $|V|$ of such operations
Then Extract_min will take total $O(V^2) +$ each edge in the Adj. List is examined, then total running time $O(V^2 + E)$
Hence $O(V^2)$ since $E$ is $O(V^2)$.

Running time

- Naïve version
  $O(|V|^2)$
- Keeping list of nodes sorted by the cost in the priority queue
  $O(|E| + |V| \log |V|)$
Graph Algorithms

- Grassfire
- Dijkstra’s
- Both explore nodes based on the distance from the start node
- They produce similar behaviors
- Both visit all or most nodes in the graph

- How can we do it more quickly

- Next A* - best first search algorithm

A*

- Extension to Dijkstra, tries to reduce the number of states
- Using heuristic estimate of the cost to go
- Evaluation function $f(n) = g(n) + h(n)$
- Operating cost function $g(n)$ – cost so far
- Heuristic function $h(n)$
  - information used to find promising node to take next
  - heuristics is admissible if it never overestimates the actual cost
A*

- Commonly used heuristic functions on grids
- Euclidean Distance

\[ H(x_n, y_n) = \sqrt{(x_n - x_g)^2 + (y_n - y_g)^2} \]

- Manhattan Distance

\[ H(x_n, y_n) = |(x_n - x_g)| + |(y_n - y_g)| \]

A* Start => A => E => goal
Nodes with higher or equal priority than goal can be pruned away. There can still be shorter path through the remaining nodes.
Keep on expanding nodes with the priority level lower than goal
Path: Start => C => K => Goal

Example: Heuristic Function

\[ h(x) \]
A*

- Complete provided finite boundary condition
- Optimal in terms of path cost
- Memory inefficient
- Exponential growth of search space with respect to the length of the solution
- How can we use it in partially known, or dynamically changing environment D*
  
  - Same as Dijkstra – different way of updating the cost of each node
  - As long as heuristics under-estimates the cost to go – A* is optimal
  - In many practical problems it is hard to find good heuristics

Planning Problems on Graphs

- Various performance measures of search:
  - Optimality, completeness, time and space complexity

- Uninformed search – blind no information is gathered from the environment
  - wavefront algorithm, BFS, DFS
- Informed search
  - some evaluation function is used, Dijkstra, A*, D*
Example: Pancake Problem

Cost: Number of pancakes flipped

50

Example: Pancake Problem

Cost: Number of pancakes flipped

51
General Tree Search

States and Actions:
- State: [1, 2, 3, 4]
- Actions: flip top two, flip all four
- Cost: 2, 4

Path to Reach Goal:
- Flip four, flip three
- Total Cost: 7

Example: Pancake Problem

State space graph with costs as weights
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place

\[ h(x) \]