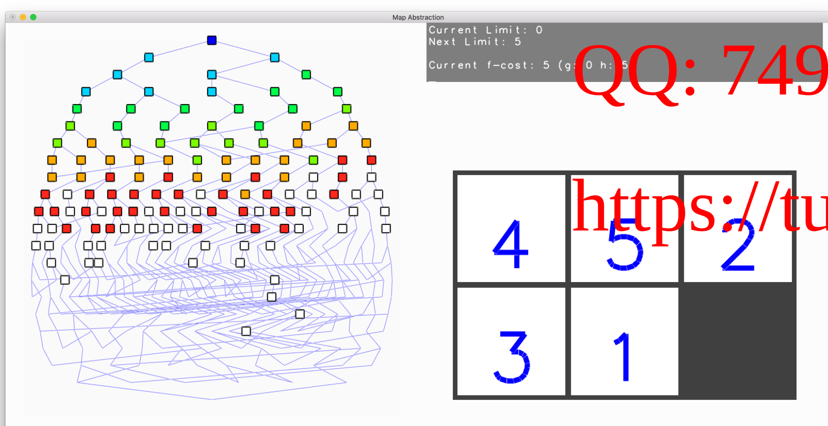
Single Agent Se

Lecture 5
IDA*, Pattern Databases

IDA*

IDA* Demo



Example

- [0 1 2 | 3 4 5]
- [0 1 2 | 4 3 5]

- Admissible heuristic

Termination (2)

- All path costs are strictly increasing
- All nodes with a given cost are exp iteration
  - Cost-limit strictly increasing
- At least 1 new node expanded eac
- No infinite-length paths of finite cos
- Must eventually expand the goal

Optimality

- Frontier -- nodes which have been generated but not expanded
  - Frontier always contains node on optimal path to goal
- Cost thresholds are monotonically increasing
- No thresholds > optimal path length
  - f(n) < optimal solution cost
- Goal has f(n) = g(n) -- no shorter solution
  - Cannot run with a threshold > g(goal)

Space Complexity

- Assume goal has cost c, minimum edge cost e
- Maximum depth of c/e (+1 for expanding at this depth)
- *e* is constant, so space is O(c)

Node Expansions

- How much work on last iteration of IDA*?
  - Same set of nodes as A*
    - Except for tie-breaking

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Node Expansions

- How much time in previous iterations?
  - Assume that the number of node
    cost *x* is N(*x*)
  - Then we usually assume N(x)/N(
  - The number of nodes grows exp
    factor of *b*) with each iteration
  - DFID analysis applies

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Node Expansions (2)

- Worst-Case performance?
  - 1 more node expanded each step
  - 1, 2, 3, … b^d - 1, b^d = O(b^{2d})

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Other Limitations of IDA*

- A* expands every state with cost < c
- IDA* expands every *node* with cost < c
  - eg. turns a graph into a tree -- doesn't detect duplicates
- What problems will IDA* work well in?
  - Sliding tile puzzle -- few cycles
- What problems will it not work well in?
  - Pathfinding -- √2 edge costs and lots of cycles

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

IDA* Demo

- http://movingai.com/SAS/IDA/
- What is the shortest cycle?
  - Find cycles in the graph
- Find an example where IDA* visits a state by two different paths

Pattern Databa

Sources of Heuristics

- Modern sources of heuristics:
  - Pattern Databases
  - True-Distance Heuristics
- Where do these work well?
- Where don't they work?

Heuristics as Relaxations

- Consider TSP - In solution:
  - All cities must be included in path
  - Each city must have two incident edges
  - Graph must be connected
- What happens if we relax/remove condition…
  - Use a MST
  - Solve sub-problems independently

Heuristics as Relaxations

- Consider route finding on a map<=>graph
  - Must travel edges
  - Otherwise just go straight to goal (euclidean)
- Another example?

Logic representation

- International planning competitions represent problems in generic language(s)
- STRIPS (Stanford Research Institu Solver) – became name of descript
  - Preconditions -- things that must an action
  - Postconditions (effects) -- how th changes when an action is applied
  - represented as add and delete lists

2x2 sliding tile puzzle

- adjacent([0, 0], [0, 1])
- adjacent([0, 0], [1, 0])
- adjacent([1, 0], [1, 1])
- adjacent([0, 1], [1, 1])
- at([1, 0], 2)
- at([0, 1], 1)
- at([1, 1], 0)
- at([0, 0], 3)

Goal

- at([0, 0], 0)
- at([1, 0], 1)
- at([0, 1], 2)
- at([1, 1], 3)

Action: Move(x, loc_1, loc_2)

- Preconditions:
  - at([loc_1], 0)
  - at([loc_2], x)
  - adjacent([loc_1], [loc_2])
- Postconditions/effects:
  - at([loc_2], 0)
  - ¬at([loc_1], 0)
  - at([loc_1], x)
  - ¬at([loc_2], x)

How do we build heuristics?

- First method:
  - Relax preconditions & solve exac
  - What happens if we relax:
    - at([loc_1], 0)?
    - at([loc_2], x)?
    - adjacent([loc_1], [loc_2])?

How do we build heuristics?

- Second method?
  - Ignore “delete” effects of postconditions
  - What happens to state?
    - Tile can be in multiple positions
  - Apply all possible moves at each step

Properties

- Will these methods produce admissible heuristics?
  - Consider that the search space is a graph
  - These methods add edges to the graph
  - Never remove edges
- Therefore, the result must be an admissible heuristic

Abstraction

- One generalized type of abstraction is one where edges are added into the search space (S)
  - Form an “edge supergraph” (T)
  - T contains all the edges in S plus possibly additional edges

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Valtorta's Theorem

- Theorem: If T is an edge super graph of S, and distances in T are computed by BF[illegible] distances in T as its heuristic is use[illegible] problem P, then for any s∈S that is [illegible] expanded if BFS is used to solve P[illegible]
  - s is expanded by A* in S, or
  - s is expanded by BFS in T
  - (BFS is reverse search)

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

How can we make this work

- Possibilities:
  - Pre-compute abstraction values
  - Decompose the heuristic computation
  - Use a different type of abstraction

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Review

- Valtorta's Theorem
  - Every node expanded by BFS in the original graph will be expanded by either the BFS in the supergraph or by A* in the original graph
- Let ϕ be a mapping from states to abstract states
  - ϕ should be a surjective function

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Generalized Valtorta's Theorem (Holte)

- If ϕ(S) is any abstraction of S, for any s∈S that is necessarily expanded if BFS is used to solve problem P, if A* is used to solve P using distances in ϕ(S) computed by BFS as its heuristic, then either:
  - s is expanded by A* in S, or
  - ϕ(s) is expanded by BFS in ϕ(S)

How do we get savings?

- If a large number of states are mapped into a single abstract state, there is a larg search in the abstract state space
  - We only have to touch 1 node in space instead of many nodes