Node Expansions

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

Node Expansions

• How much time in previous iterations?
  • Assume that the number of nodes expanded of cost \(x\) is \(N(x)\)
  • Then we usually assume \(N(x)/N(x-1) = b\)
  • The number of nodes grows exponentially (by a factor of \(b\)) with each iteration
  • DFID analysis applies

Node Expansions (2)

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

### DFBnB

- DFS for solving problems
  - Usually when we have a bounded solution length
    - eg TSP, assignment (K items N to groups)
  - Maintain cost of best solution so far
  - Will explore solutions with cost > $c$
  - Iteratively improve solution cost

### DFBnB

- Maintain: partial solution cost ($g$), underestimate of remaining solution ($h$)
  - Worthwhile to compute an expensive heuristic (eg MST)
  - Can be used as an any-time algorithm
  - Node ordering improves efficiency
    - eg in TSP take the shortest path next
  - Halt when all other solutions shown suboptimal

DFBnB(start, path, &best)

for each successor $s_i$ of start

if ($\text{cost}(\text{path}+s_i) + h(\text{path}+s_i) > \text{best}$)
    continue

DFBnB($s_i$, path+$s_i$, best)
if no successors
    $\text{cost} \leftarrow \text{path-cost(path)}$
if ($\text{cost} < \text{best}$)
    $\text{best} \leftarrow \text{cost}$
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

Logic representation

• International planning competitions represent problems in generic language(s)

• STRIPS (Stanford Research Institute Problem Solver) – became name of description language

• Preconditions -- things that must be true to apply an action
• Postconditions (effects) -- how the state changes when an action is applied
• represented as add and delete lists

Heuristics as Relaxations

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

• Another example?
2x2 sliding tile puzzle

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

```
0 1
2 3
```

Goal

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

```
0 1
3 2
2 3
1 0
```

Action: Move(x, loc₁, loc₂)

- Preconditions:
  - at([loc₁], 0)
  - at([loc₂], x)
  - adjacent([loc₁], [loc₂])

```
0 1
2 3
```

- Postconditions:

How do we build heuristics?

- First method:
  - Relax preconditions & solve exactly
  - What happens if we relax:
    - at([loc₁], 0)?
    - at([loc₂], x)?
    - adjacent([loc₁], [loc₂])?
How do 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
  • Demonstrate

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

Generalized abstraction

• Theorem: If T is an edge super graph of S, and distances in T are computed by BFS, and A* with distances in T as its heuristic is used to solve problem P, then for any s ∈ S that is necessarily expanded if BFS is used to solve P, either:
  • s is expanded by A* in S, or
  • s is expanded by BFS in T
  • (BFS is reverse search)
How can we make this work

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

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 \( \phi \) be a mapping from states to abstract states
  - \( \phi \) should be a surjective function

Generalized Valtorta’s Theorem (Holte)

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

How do we get savings?

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

- Take states and replace some values with blanks / colors
  - (0 1 2 3 4 5 6 7 8 9)
  - (0 1 2 3 4 5 6 7 8 9)
  - (0 * 2 * 4 * 6 * 8 *)
- Extreme example
  - (0 * * * * * * * *)
- \( \phi(S) \) is this mapping function

PDB Idea

- Apply a domain abstraction
  - In the abstract state space, perform a BFS from the goal
  - To get a heuristic from state \( s \), apply domain abstraction to \( s \) and lookup cost in BFS
  - Need an efficient way to store and lookup results of BFS

Examples

- Pancake Puzzle
- Towers of Hanoi
- Top Spin

Why does this work

- If the problem space grows as \( b^d \)
  - The solution length grows with \( d \)
  - Suppose \( w \) is the state space width
  - Uniformly abstract \( k \) states together
  - What is the new width?
  - \( b^h = b^w/k \)
Maximum heuristic after abstraction

- $b^h = b^w/k$
- $\log(b^h) = \log(b^d) - \log(k)$
- $h \cdot \log(b) = d \cdot \log(b) - \log(k)$
- $h = d - \log(k)/\log(b)$
- $h = d - \log_b(k)$

Methodology

- Pattern Database
- Precomputation of values
- Single goal state
- BFS from goal in abstract state space