Artificial Intelligence Programming

Heuristic Search

Chris Brooks

Department of Computer Science

University of San Francisco

6-2: Overview

- Heuristic Search - exploiting knowledge about the problem
- Heuristic Search Algorithms
    - “Best-first” search
    - Greedy Search
    - A* Search
    - Extensions to A*
- Constructing Heuristics

6-3: Informing Search

- Previous algorithms were able to find solutions, but were very inefficient.
  - Exponential number of nodes expanded.
- By taking advantage of knowledge about the problem structure, we can improve performance.
- Two caveats:
  - We have to get knowledge about the problem from somewhere.
  - This knowledge has to be correct.

6-4: Best-first Search

- Recall Uniform-cost search
  - Nodes were expanded based on their total path cost
  - A priority queue was used to implement this.
- Path cost is an example of an *evaluation function*.
  - We'll use the notation $f(n)$ to refer to an evaluation function.
- An evaluation function tells us how promising a node is.
- Indicates the quality of the solution that node leads to.

6-5: Best-first Search

- By ordering and expanding nodes according to their f value, we search the “best” nodes “first”
- If f was perfect, we would expand a straight path from the initial state to the goal state.
  - Arad, Sibiu, Rimnicu Vilcea, Pitesti, Bucharest
- Of course, if we had a perfect f, we wouldn’t need to solve the problem in the first place.
- Instead, we’ll try to develop heuristic functions $h(n)$ that help us estimate $f(n)$.

6-6: Best-first Search

- Best-first Pseudocode

```
enqueue(initialState)
do
  node = dequeue()
  if goalTest(node)
     return node
  else
    children = successor-fn(node)
    for child in children
        insert-with(child, f(child))
```

- where insert-with orders our priority queue accordingly.

6-7: Greedy Search

- Let's start with the opposite of uniform-cost search
- UCS used the solution cost to date as an estimate of f
- *Greedy search* uses an estimate of distance to the goal for f.
- Rationale: Always pick the node that looks like it will get you closest to the solution.
- Let's start with a simple estimate of f for the Romania domain.
  - $h(city) =$Straight-line distance between that city and Bucharest.

6-8: Greedy Search

- Notice that there wasn't anything in the problem description about straight-line distance or the fact that that would be a useful estimate.
- We used our knowledge about the way roads generally work.
- This is sometimes called *common sense* knowledge.

6-9: Greedy Search Example

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- A = 336
- (dequeue A) S = 253, T = 329, Z = 374
- (dequeue S) F = 176, RV = 193, T = 329, A = 336, Z = 374, O = 380
- (dequeue F) B = 0, RV = 193, S = 253, T = 329, A = 336, Z = 374, O = 380
- dequeue B - solution: A -> S -> F ->B

- We found a solution quickly, but it was not optimal.
- What was the problem with our approach?

6-10: Issues with Greedy Search

- Sometimes the optimal solution to a problem involves moving ’away’ from the goal.
- For example, to solve 8-puzzle, you often need to ‘undo’ a partial solution.
- Greedy search has many of the same appeals and weaknesses as DFS.
  - Expands a linear number of nodes
  - Is not complete or optimal
- Its ability to cut toward a goal is appealing - can we salvage this?

6-11: A* search

- A* search is a combination of uniform cost search and greedy search.
- A node's $f(n) = g(n) + h(n)$
  - $g(n)$ = current path cost
  - $h(n)$ = heuristic estimate of distance to goal.
- Favors nodes that have a cheap solution to date and also look like they'll get close to the goal.
- If $h(n)$ satisfies certain conditions, A* is both complete (always finds a solution) and optimal (always finds the best solution).

6-12: A* example - Romania

- h = straight-line distance

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- Arad = 0 + 366 = 366
- (dequeue A: g = 0) S = 140 + 253 = 393, T = 118 + 329 = 447, Z = 75 + 374 = 449
- (dequeue S: g = 140) RV = 220 + 193 = 413, F = 239 + 176 = 415, T = 118 + 329 = 447, Z = 374 + 75 = 449, A = 280 + 336 = 616, O = 291 + 380 = 671,
- (dequeue RV: g = 220) F = 239 + 176 = 415, P = 317 + 100 = 417, T = 118 + 329 = 447, Z = 374 + 75 = 449, C = 366 + 160 = 526, S = 300 + 253 = 553, A = 280 + 336 = 616, O = 291 + 380 = 671
- (dequeue F: g = 239) P = 317 + 100 = 417, T = 118 + 329 = 447, Z = 374 + 75 = 449, C = 366 + 160 = 526, B = 550 + 0 = 550, S = 300 + 253 = 553, S = 338 + 253 = 591,A = 280 + 336 = 616, O = 291 + 380 = 671

6-13: A* example - Romania

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- (dequeue P: g = 317) T = 118 + 329 = 447, Z = 374 + 75 = 449, B = 518 + 0 = 518, C = 366 + 160 = 526, B = 550 + 0 = 550, S = 300 + 253 = 553, S = 338 + 253 = 591, RV = 414 + 193 = 607, C = 455 + 160 = 615, A = 280 + 336 = 616, O = 291 + 380 = 671
- (dequeue T: g = 118) Z = 374 + 75 = 449, L = 229 + 244 = 473, B = 518 + 0 = 518, C = 366 + 160 = 526, B = 550 + 0 = 550, S = 300 + 253 = 553, A = 236 + 336 = 572, S = 338 + 253 = 591, RV = 414 + 193 = 607, C = 455 + 160 = 615, A = 280 + 336 = 616, O = 291 + 380 = 671
- (dequeue Z: g = 75) L = 229 + 244 = 473, A = 150 + 336 = 486, B = 518 + 0 = 518, O = 146 + 380 = 526, C = 366 + 160 = 526, B = 550 + 0 = 550, S = 300 + 253 = 553, A = 236 + 336 = 572, S = 338 + 253 = 591, RV = 414 + 193 = 607, C = 455 + 160 = 615, A = 280 + 336 = 616, O = 291 + 380 = 671

6-14: A* example - Romania

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

- (dequeue L: g = 229) A = 150 + 336 = 486, B = 518 + 0 = 518, O = 146 + 380 = 526, C = 366 + 160 = 526, M = 299 + 241 = 540, B = 550 + 0 = 550, S = 300 + 253 = 553, A = 236 + 336 = 572, S = 338 + 253 = 591, RV = 414 + 193 = 607, C = 455 + 160 = 615, A = 280 + 336 = 616, T = 340 + 329 = 669, O = 291 + 380 = 671
- (dequeue A: g = 150) B = 518 + 0 = 518, O = 146 + 380 = 526, C = 366 + 160 = 526, M = 299 + 241 = 540, S = 290 + 253 = 543, B = 550 + 0 = 550, S = 300 + 253 = 553, A = 236 + 336 = 572, S = 338 + 253 = 591, T = 268 + 329 = 597, Z = 225 + 374 = 599, RV = 414 + 193 = 607, C = 455 + 160 = 615, A = 280 + 336 = 616, T = 340 + 329 = 669, O = 291 + 380 = 671
- (dequeue B: g = 518) solution. A -> S -> RV -> P -> B

6-15: Optimality of A*

- A* is optimal (finds the shortest solution) as long as our h function is *admissible*.
  - Admissible: always underestimates the cost to the goal.
- Proof: When we dequeue a goal state, we see $g(n)$, the actual cost to reach the goal. If h underestimates, then a more optimal solution would have had a smaller $g + h$ than the current goal, and thus have already been dequeued.
- Or: If h overestimates the cost to the goal, it's possible for a good solution to “look bad” and get buried further back in the queue.
- In our Romania example, SLD always underestimates.

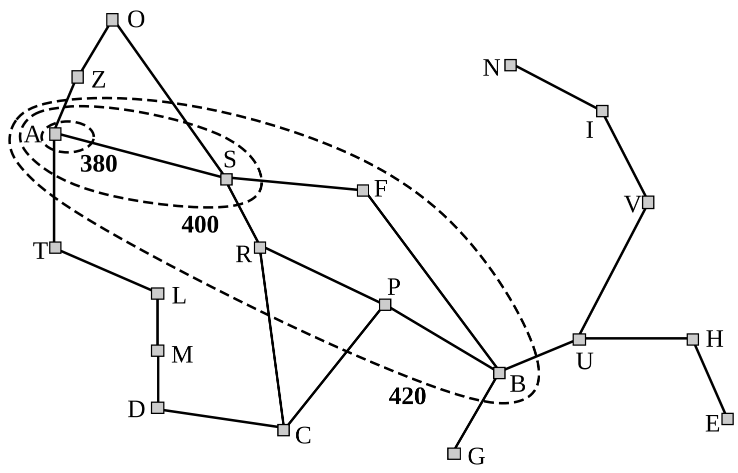
6-16: Optimality of A*

- Notice that we can't discard repeated states.
  - We could always keep the version of the state with the lowest g
- More simply, we can also ensure that we always traverse the best path to a node first.
- a *monotonic* heuristic guarantees this.
- A heuristic is monotonic if, for every node n and each of its successors (n'), $h(n)$ is less than or equal to $stepCost(n, n') + h(n')$.
  - In geometry, this is called the triangle inequality.

6-17: Optimality of A*

- SLD is monotonic. (In general, it's hard to find realistic heuristics that are admissible but not monotonic).
- Corollary: If h is monotonic, then f is nondecreasing as we expand the search tree.
- Alternative proof of optimality.
- Notice also that UCS is A* with $h(n) = 0$
- A* is also *optimally efficient*
  - No other complete and optimal algorithm is guaranteed to expand fewer nodes.

6-18: Pruning and Contours



- Topologically, we can imagine A* creating a set of contours corresponding to f values over the search space.
- A* will search all nodes within a contour before expanding.
- This allows us to *prune* the search space.
  - We can chop off the portion of the search tree corresponding to Zerind without searching it.

6-19: Building Effective Heuristics

- While A* is optimally efficient, actual performance depends on developing accurate heuristics.
- Ideally, h is as close to the actual cost to the goal (h^*)as possible while remaining admissible.
- Developing an effective heuristic requires some understanding of the problem domain.

6-20: Effective Heuristics - 8-puzzle

- h1 - number of misplaced tiles.
  - This is clearly admissible, since each tile will have to be moved at least once.
- h2 - *Manhattan distance* between each tile’s current position and goal position.
  - Also admissible - best case, we’ll move each tile directly to where it should go.
- Which heuristic is better?

6-21: Effective Heuristics - 8-puzzle

- h2 is better.
  - We want h to be as close to h^* as possible.
- If $h2(n) > h1(n)$ for all n, we say that h2 *dominates* h1.
- We would prefer a heuristic that dominates other known heuristics.

6-22: Finding a heuristic

- So how do we find a good heuristic?
- Solve a relaxed version of the problem.
  - Cost of an optimal solution to the relaxed version is an admissible heuristic for the original problem.
  - 8-puzzle - allow tiles to move over each other.
  - Romania - assume that there is a road from every city to the goal.
- Solve subproblems
  - Cost of moving one tile, traversing a path
  - Often, these subproblems (and their solutions) are then cached.
- Learning from experience

6-23: Improving A*

- A* has one big weakness - Like BFS, it potentially keeps an exponential number of nodes in memory at once.
- Iterative deepening A* is a workaround.
- Rather than searching to a fixed depth, we search to a fixed f-cost.
  - If the solution is not found, we increase f and start again.
- Works in worlds with uniform, discrete-valued step costs

6-24: Improving A*

- Recursive best-first search
  - Combination of DFS and A*.
  - Do DFS, but keep the f-cost of all fringe nodes.
  - If expansion leads to a node worse than something in the fringe, backtrack.
- Improvement over A*, but not spectacular.
- Both IDA* and RBFS throw away too much.

6-25: Improving A*

- SMA*
- Regular A*, plus a fixed limit on memory used.
- When memory is full, discard the node with the highest f.
- Value of discarded node is assigned to the parent.
  - This allows SMA* to 'remember' the value of that branch.
  - If all other branches get a higher f value, this child will be regenerated.
- SMA* is complete and optimal.
- On very hard problems, SMA* can wind up repeatedly deleting and regenerating branches.
  - Moral: Often, memory requirements make our problem intractable before time requirements.

6-26: Summary

- Problem-specific heuristics can improve search.
- Greedy search
- A*
- Developing heuristics
  - Admissibility, monotonicity, dominance
- Memory issues

6-27: Next time ...

- Constraint Satisfaction
  - Scheduling
  - Map coloring
  - Satisfiability
- Branch-and-bound search