EECS 3401 — AI and Logic Prog. — Lecture 12

Adapted from slides of Yves Lesperance

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

November 2, 2020

- Today: **Search Algorithms Continued**
- Required reading: Russell & Norvig Chapters 3.1–3.6, 4.1

Depth-Limited Search

- Breadth-first search has computational problems (esp. space). Depth-first can run off down a very long (infinite) path.
- **Depth-Limited Search**
  - Perform DFS but only to a pre-specified depth limit L
  - No node on a path that is more than L steps from the initial state is placed on the Frontier
  - We truncate the search by looking only at paths of length L or less
- Infinite-length paths are no longer a problem!
- But will only find a solution if a solution of length $\leq L$ exists

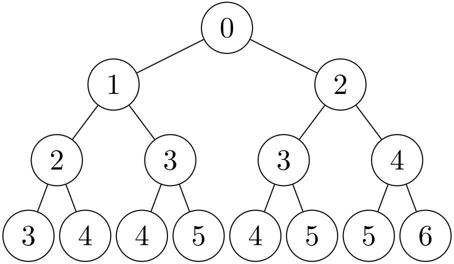
Depth-Limited Search

```
DLS(Frontier, Successors, Goal?)
    If Frontier is empty:
        return Failure
    Curr := select state from Frontier
    If Goal?(Curr):
        return Curr
    If Depth(Curr) < L:
        NewFrontier := (Frontier - {Curr}) + Successors(Curr)
    Else:
        NewFrontier := Frontier - {Curr}
        CutoffOccurred := True
    return DLS(NewFrontier, Successors, Goal?)
```

Iterative Deepening Search

- Take the idea of DLS one step further
- Starting at depth limit $L = 0$, we iteratively increase the depth limit, performing a depth-limited search for each depth limit
- Stop if no solution is found or if the depth limited search failed without cutting off any nodes becayse of the depth limit

Iterative Deepening Search: Example



Iterative Deepening Search Properties

Completeness

- Yes, if solution of length d exists, it will be found when $L = d$

Time Complexity

- At first glance, looks bad because nodes are expanded many times
- $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
  Root expanded $d+1$ times, level 1 nodes expanded d times, etc.
- Example: $b = 4, d = 10$
  - $11 \cdot 4^0 + 10 \cdot 4^1 + 9 \cdot 4^2 + \ldots + 2 \cdot 4^9 = 815555$
  - $4^{10} = 1048576$
  - Most nodes lie on the bottom layer
  - In fact, IDS can be more efficient than breadth-first search: nodes at limit are not expanded. BFS must expand all nodes until it expands a goal node.

Iterative Deepening Search Properties

Space Complexity

- $O(bd)$ — still linear

Optimality

- Will find shortest-length solution (which is optimal if costs are uniform)
- If costs are not uniform, we can use a "cost" bound instead
  - Only expand paths of cost less than the cost bound
  - Keep track of the minimum cost unexpanded path in each depth-first iteration, increase the cost bound to this on the next iteration
  - This can be very expensive. Need as many iterations of the search as there are distinct path costs

A Note on Cycle Checking

- Idea: Keep track of **all states** previously expanded during the search
- When we expand node n_k to obtain child c, ensure c is not equal to any previously expanded state
- This is known as *cycle checking* or *multiple path checking*
- Why can't we utilize this technique with DFS? — what happens to space complexity?
- Thus, only useful with BFS, which is already bad in terms of space complexity

Heuristic Search

- In uninformed search, we don't try to evaluate which of the nodes on the frontier are the *most promising*. We never *look-ahead* to the goal
- Even with uniform-cost search, we always expand the cheapest path regardless of what and where the goal is.
- Often, we have some other knowledge about the merit of nodes, e.g., going the wrong direction in Romania

Heuristic Search

Different notions of **merit**

- If we are concerned about the **cost of the solution**, we might want a notion of merit of how costly it is to get to the goal from that search node
- If we are concerned about **minimizing computation** in search, we might want a notion of ease in finding the ggoal from that search node
- We will focus on the **cost of solution** notion of merit

Heuristic Search

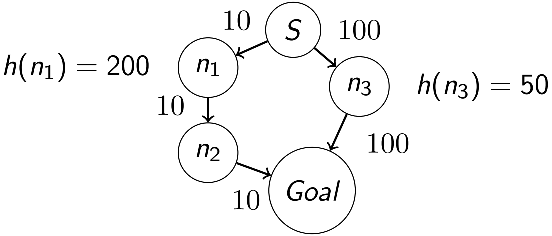
- The idea is to develop a domain-specific heuristic function $h(n)$
- $h(n)$ **guesses** the cost of getting to the goal from node n
- There are different ways of guessing this cost in different domains. That is, heuristics are *domain-specific*

Heuristic Search

- Convention: If $h(n_1) < h(n_2)$, this means that we guess that it is cheaper to get to the goal from n_1 than from n_2
- We require that $h(n) = 0$ for every node n that satisfies the goal.

Using only $h(n)$ — Greedy best-first search

- Use $h(n)$ to rank the nodes on open and always expand the node with lowest h-value
- We are greedily trying to achieve a low-cost solution
- However, this method ignores the cost of getting to n, so it ca be lead astray exploring nodes that cost a lot to get to but seem to be close to the goal



A* Search

- Take into account the cost of getting to the node as well as our estimate of the cost of getting to the goal from n
- Define: $f(n) = g(n) + h(n)$, where
  - $g(n)$ is the cost of the path to node n
  - $h(n)$ is the heuristic estimate of the cost of getting to a goal node from node n
- **Always expand the node with lowest f-value on the frontier**
- The f-value is an estimate of the cost of getting to the goal via this node (path)

Conditions on $h(n)$

- We want to analyze the behaviour of the resultant search
- Completeness, time, space, optimality?
- To obtain such results, we must put some further conditions on the heuristic function $h(n)$ and the search space

Conditions on $h(n)$: Admissibility

- Assume as usual that $c(n_1 \to n_2) \geq \epsilon > 0$ — the cost of any transition is greater than zero and can't be arbitrarily small
- Let $h^*(n)$ be the cost of an optimal path fron n to a goal node (∞ if there is no path)
- A heuristic h is **admissible** if it satisfies the condition

$$h(n) \leq h^*(n).$$

That is, an admissible h always *underestimates* the true cost, never overestimates.

Consistency / monotonicity

- A heuristic h is **monotone** (**consistent**) if it satisfies the trangle inequality for all nodes n_1, n_2:

$$h(n_1) \leq c(n_1 \rightarrow n_2) + h(n_2).$$

- Note that there might be multiple transitions (action) from n_1 to n_2, and the inequality must hold for all of them
- This is a stronger condition than admissibility. Monotonicity implies admissibility.

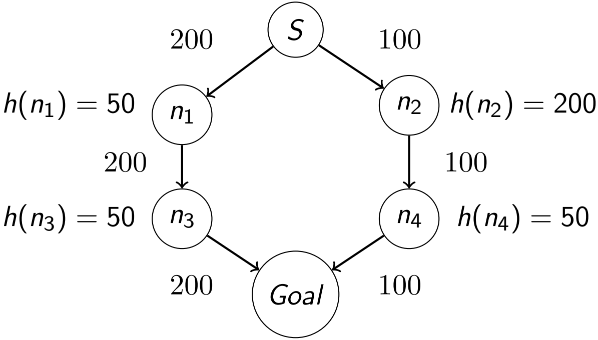
Intuition behind admissibility

- $h(n) \leq h^*(n)$ means that the search won't miss any promising paths
- If it really is cheap to get to a goal via n (i.e., both $g(n)$ and $h^*(n)$ are low), then $f(n) = g(n) + h(n)$ will also be low, and the search won't ignore n in favour of more expensive options
- This can be formalized to show that admissibility implies optimality

Intuition behind monotonicity

- $h(n_1) \leq c(n_1 \rightarrow n_2) + h(n_2)$
- This says something similar, but in addition one won't be "locally" mislead — see example

Example: admissible but non-monotonic



$$
\begin{aligned}
\{S\} &\Rightarrow \{n_1[200 + 50 = 250], n_2[200 + 100 = 300]\} \\
&\Rightarrow \{n_2[200 + 100 = 300], n_3[400 + 50 = 450]\} \\
&\Rightarrow \{n_4[200 + 50 = 250], n_3[400 + 50 = 450]\} \\
&\Rightarrow \{Goal[300 + 0 = 300], n_3[400 + 50 = 450]\}
\end{aligned}
$$

We do find the optimal path as the heuristic is still admissible, but we are mislead into ignoring n_2 until after we expand n_1

Monotonicity implies admissibility

$$\text{If} \quad h(n_1) \leq c(n_1 \to n_2) + h(n_2) \quad \text{then} \quad h(n) \leq h^*(n)$$

Proof by induction on the number of steps to a goal node M

- (Base case) If n is a goal node, then $h(n) = 0 = h^*(n)$, so $h(n) \leq h^*(n)$
- (Induction step) Assume $h(n_k) \leq h^*(n_k)$ if number of steps to goal at n_k is at most K. Show that the proposition must hold for nodes n_{k+1} where number of steps to goal is $K + 1$:
  - Let n_k be the next node along a shortest path from n_{k+1} to goal
  - Since h is monotone, have $h(n_{k+1}) \leq c(n_{k+1} \to n_k) + h(n_k)$
  - By ind. hypothesis, $h(n_k) \leq h^*(n_k)$
  - So $h(n_{k+1}) \leq c(n_{k+1} \to n_k) + h^*(n_k)$
  - Thus, $h(n_{k+1}) \leq h^*(n_{k+1})$
- If goal is unreachable from a node n, then $h^*(n) = \infty$ and the result trivially holds.

Consequences of monotonicity

1. **The f-values of nodes along a path must be non-decreasing**
   - I.e., If path is $\langle S \to n_1 \to n_2 \to \ldots \to n_k \rangle$, we claim that $f(n_i) \leq f(n_{i+1})$
   - Proof:

$$
\begin{aligned}
f(n_i) &= c(Start \to \ldots \to n_i) + h(n_i) \\
&\leq c(Start \to \ldots \to n_i) + c(n_i \to n_{i+1}) + h(n_{i+1}) \\
&= c(Start \to \ldots \to n_i \to n_{i+1}) + h(n_{i+1}) \\
&= g(n_{i+1}) + h(n_{i+1}) \\
&= f(n_{i+1})
\end{aligned}
$$

Consequences of monotonicity

2. If n_2 is expanded after n_1, then $f(n_1) \leq f(n_2)$

- Proof:
  - If n_2 was on the frontier when n_1 was expanded, then $f(n_1) \leq f(n_2)$, because otherwise we would've selected n_2 to expand
  - If n_2 was added to the frontier after n_1's expansion, then let n be an ancestor of n_2 that was present when n_1 was being expanded (this could be n_1 itself). We have $f(n_1) \leq f(n)$ since A* chose n_1 rather than n. Also, since n is along the apth to n_2, by the previous property, we have $f(n) \leq f(n_2)$. Thus, we get $f(n_1) \leq f(n_2)$.

Consequences of monotonicity

3. When n is expanded, every path with a lower f-value has already been expanded

- Assume by contradiction that there exists a path $\langle Start, \ldots, n_{i-1}, n_i, n_{i+1}, \ldots, n_k \rangle$ with $f(n_k) < f_n$ and n_i is its last expanded node.
- Then n_{i+1} must be on the frontier while n is expanded.
  1. By (1), $f(n_{i+1}) \leq f(n_k)$ since they lie along the same path
  2. Since $f(n_k) < f(n)$, we get $f(n_{i+1}) < f(n)$
  3. By (2), $f(n) \leq f(n_{i+1})$ since n is expanded before n_{i+1}

  Contradiction in last two points.

Consequences of monotonicity

4. With a monotone heuristic, the first time A* expands a state, it has found the minimum cost path to that state

Proof:

- Let $Path_1$ be $\langle Start, n_0, \ldots, n_k, n \rangle$ be the **first** path to n found. Then $f(Path_1) = c(Path_1) + h(n)$.
- Let $Path_2$ be $\langle Start, m_0, \ldots, m_j, n \rangle$ be the **another** path to n found later. Then $f(Path_2) = c(Path_2) + h(n)$.
- By property (3), $f(Path_1) \leq f(Path_2)$
- Hence, $c(Path_1) \leq c(Path_2)$

Consequences of monotonicity

Completeness

- Yes
- Consider a least-cost path to goal: $Solution = \langle Start, n_0, \ldots, Goal \rangle$ with cost $c(Solution)$
- Since each action has a cost $\geq \epsilon > 0$, there are only a finite number of nodes (paths) that have cost $\leq c(Solution)$
- All of these paths myst be explored before any path of cost $> c(Solution)$
- So eventually *Solution*, or some equal-cost path to a goal must be expanded

Time and Space Complexity

- When $h(n) = 0$ for all n, h is monotone, and A* becomes uniform-cost search.
- It can be shown that when $h(n) > 0$ for some n, the number of nodes expanded can be no larger than uniform-cost
- Thus, same worst-case bounds as uniform-cost apply

Consequences of monotonicity

Optimality

- Yes, by last property, the first path to a goal node must be optimal

Cycle Checking

- If we do cycle checking, it is still optimal. Due to last property, we need to keep only the first path to a node, rejecting all subsequent paths

End of Lecture

- Next time: **Heuristic Search continued**