Lecture 3: Informed (Heuristic) Search

Viliam Lisý & **Branislav Bošanský**

Artificial Intelligence Center
Department of Computer Science, Faculty of Electrical Eng.
Czech Technical University in Prague

bosansky@fel.cvut.cz

March, 2024

Formalization

Last week – formalization:

1. a well-defined problem (inputs / outputs for the algorithm)
2. formalization helps to think about the problem (e.g., formalizing the dynamics)
3. formalization is independent on the algorithm (problems formalized using a standard formalism can be solved with different algorithms)

Uninformed / Informed search algorithms solve deterministic MDPs (but they can be solved also with learning-based approaches).

Search Algorithm – Uniform-Cost Search

1. Start from the initial state S_0
2. Apply available actions to the current state and generate new possible states
3. Select one of the newly generated states as the current one
4. If the current state is the goal state $\rightarrow$ finish
5. If not, go to step 2

uniform-cost search $\rightarrow$ the state that accumulated the lowest cost is selected for the expansion

Question

For the uniform-cost search (when we optimize the costs rather the number of actions), can we terminate the search when the goal state is generated (added into the open list)?

Improving Uniform-Cost Search

Question

For the uniform-cost search (when we optimize the costs rather the number of actions), can we terminate the search when the goal state is generated (added into the open list)?

NO $\rightarrow$ the algorithm can terminate only when the goal state is selected for expansion (it is the state with lowest costs)

The lowest accumulated cost does not necessarily mean that the state will be on the optimal path to the goal.

What is a good predictor of the future? $\rightarrow$ heuristics

Heuristics

Heuristics

Estimate of the cost from the current state to the goal. Denoted $h : S \to \mathbb{R}^+$

We can use the heuristics estimate to choose next states to expand.

Optimal heuristics h^* – the optimal cost from a state to goal.

In practice, we want to be as close to the optimal heuristics as possible. The estimate is typically a solution of a simplified problem.

We can order states in fringe according to the heuristic value.

How would that work? Will the algorithm always find an optimal solution?

Using Heuristics

```
# # # # # #
# G       #
# # # #   #
#   # #   #
#   @     #
# # # # # #
```

What can be a good heuristic for the problem of collecting the gold in a maze?

- Manhattan distance from the current position to the gold
- Euclidean distance
- ...

Using Heuristics – Greedy Best-First Search

What if we use the Manhattan distance? The heuristic values are as follows:

```
#  #  #  #  #  #
#  G  1  2  3  #
#  #  #  #  4  #
#  2  #  #  5  #
#  3  @  5  6  #
#  #  #  #  #  #
```

Obviously, following the heuristics in a greedy manner will result in a suboptimal path.

If we do not maintain the closed list, the greedy best-first search algorithm will not terminate!

Using Heuristics – Greedy Best-First Search

We know that uniform-cost search algorithm works here

```
#  #  #  #  #  #
#  G  7  6  5  #
#  #  #  #  4  #
#  2  #  #  3  #
#  1  @  1  2  #
#  #  #  #  #  #
```

Accumulated costs can help the algorithm to get out of parts of the state space that the heuristics incorrectly evaluates as promising.

What if we select the next state to expand based on the sum of accumulated costs and heuristic estimation?

Using Heuristics (2)

```
#    #    #    #    #    #      #    #    #    #    #    #
#    G                   #      #    G                   #
#    #    #    #         #      #    #    #    #         #
#         #    #         #      #   2+2   #    #         #
#   1+3   @   1+5        #      #    @   2+4  1+5        #
#    #    #    #    #    #      #    #    #    #    #    #

#    #    #    #    #    #      #    #    #    #    #    #
#    G                   #      #    G                   #
#    #    #    #         #      #    #    #    #         #
#    @    #    #         #      #         #    #         #
#   3+3  2+4  1+5        #      #   3+3  2+4   @   2+6   #
#    #    #    #    #    #      #    #    #    #    #    #
```

...

Correct Use of Heuristics - A* Search

Let $s \in S$ be the current state represented as a node $n \in N$ in the search tree. Now, $g : N \rightarrow \mathbb{R}^+$ is the cost accumulated on the path from the starting state to the current state s along the path in the search tree to node n. In the A^* algorithm, the choice of the next node to expand is the one that minimizes the function

$$f(n) = g(n) + h(n)$$

Do we have guarantees that such algorithm finds an optimal solution?

Correct Use of Heuristics - Example

What if we have a heuristic function as follows?

#	#	#	#	#	#
#	*G*	0	0	0	#
#	30	#	#	0	#
#	20	#	#	0	#
#	10	@	0	0	#
#	#	#	#	#	#

The right-hand (and longer) path would be found first. $\rightarrow$ If we use an arbitrary heuristic function, the algorithm is not optimal.

How do we guarantee the optimality? $\rightarrow$ The algorithm has to use "meaningful heuristics".

Heuristics Properties – Admissibility

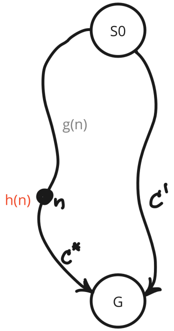
Definition

A heuristic is **admissible** if it never overestimates the cost to the goal (the heuristic is always optimistic; $h(n) \leq h^*(n)\ \forall n \in N$).

Theorem

If the heuristic is **admissible**, A* is always optimal (finds an optimal solution).

Optimality of A*



proof

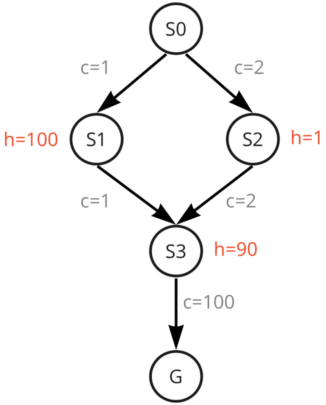
Let C^* be the cost of the optimal path to goal state G and let's assume that the A^* algorithm found a suboptimal path with cost $C' > C^*$.

In the open list of the algorithm, there must be a node that is on the optimal path from the starting state to the goal (denote it n).

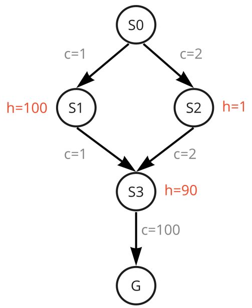
$f(n) = h(n) + g(n) \leq h^*(n) + g(n) \leq C^* < C'$
therefore, node n should have been selected for the expansion before goal state G reached via suboptimal path.

Optimality of A*

Is this enough? What if the algorithm reaches the same state multiple times ... can we discard the next visits or does the algorithm need to re-evaluate already closed states?



Optimality of A*



The algorithm explores:

- s_2; $f(s_2) = 2 + 1 \rightarrow s_3$ added to fringe
- s_3; $f(s_3) = 4 + 90 \rightarrow G$ added to fringe
- s_1; $f(s_1) = 1 + 100 \rightarrow s_3$?

The algorithm cannot dismiss it as already solved since it found a better path to s_3. Otherwise, the algorithm would miss the optimal path $s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow G$

It is safe to discard already explored state if the cost of the new path is greater or equal than the cost with which the state has been explored.

Heuristics Properties – Consistency

Definition

Consider two nodes in the search tree n and n' such that n' is reached immediately by making an action in n. Denote $c(n, n')$ the cost for this action. A heuristic is **consistent** if the following holds:

$$h(n) \leq h(n') + c(n, n')$$

(or $g(n) + h(n) \leq g(n') + h(n')$).

Consistency is a stronger property than admissibility (every consistent heuristic is admissible).

Intuitions:

- similar to triangle inequality
- heuristic function is more informative deeper in the search tree

Optimality of A* – Efficiency

Theorem

Assume that the A^* algorithm uses a consistent heuristic. If a state is selected for exploration, the algorithm has already found an optimal path to this state.

Theorem

A^* algorithm is **optimally efficient** *in the sense no other optimal algorithm is guaranteed to expand fewer nodes than A^*.*

A^* expands all nodes in the search tree with $f(n) = g(n) + h(n) < C^*$. Any other algorithm has to explore these nodes as well, otherwise it can miss the optimal solution.

Iterative Deepening A* (IDA*)

Despite the theoretical properties, memory requirements of the A* algorithm can be still significant (note that $h(n) = 0$ for all nodes n in the search tree is a consistent heuristic).

We can use **limited-cost A*** with cost cutoff c, such that any node with $g(n) + h(n) > c$ is not expanded. **IDA*** gradually increases the cost cutoff.

Other variants:

- recursive best-first search (RBFS)
- (simplified) memory-bounded A* (MA* / SMA*)

Designing Heuristics

How should we design admissible heuristics?

- solve a simplified (relaxed) problem
  (e.g., there are no obstacles in a maze)
- solve only a single subproblem
- split into more subproblems
- ...

The heuristic function has to be informative (note that $h(s) = 0$ for all states $s \in S$ is an admissible heuristic but it is not very informative).

Consider two admissible heuristic functions h_1 and h_2 such that $h_1(s) \geq h_2(s)$ for all states $s \in S$. We say that h_1 dominates h_2 and is more informative.

More informative heuristics expand fewer nodes in the search tree.

Example Heuristics

Recall the 8-puzzle problem

1		2
4	5	3
7	8	6

What are the possible admissible heuristics?

- number of correctly placed tiles
- sum of Manhattan distances of tiles to their target location
- solving a subproblem

$$\begin{array}{|c|c|c|} \hline 1 & & 2 \\ \hline 4 & * & 3 \\ \hline * & * & * \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & * & * \\ \hline * & * & \\ \hline \end{array}$$

all such combinations can be pre-computed and stored in a database

Example Heuristics – Does it Matter?

YES

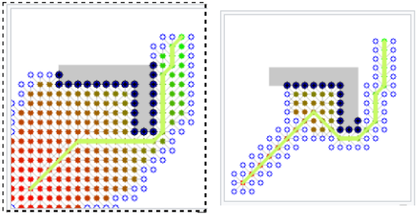
Compare the average number of expanded nodes for the number misplaced tiles (h_1) and the sum of Manhattan distances (h_2):

Sol. length	Iterative Deepening	$A^*(h_1)$	$A^*(h_2)$
8	6384	39	25
12	3644035	227	73
16	—	1301	211
20	—	7276	676

Inadmissible Heuristics

In some cases, admissible heuristics can still expand too many nodes → the algorithm is slow, requires large memory (open / closed list).

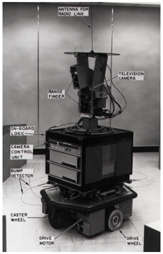
We can sacrifice optimality guarantees for performance by using inadmissible heuristics. Let $h_a(s)$ be an admissible heuristic function. If we use a modified heuristic function $h_w(s) = \varepsilon \cdot h_a(s)$ for some $\varepsilon > 1$, the found solution has a cost at most ε-times of the optimal one.



(not so) Ancient AI

One of the best known algorithms in AI.

Created as a part of the Shakey project (1968)



But the research has not stopped with A* $\rightarrow$ many variants.

AAAI 2016 Best Paper Award $\rightarrow$ **Bidirectional Search That Is Guaranteed to Meet in the Middle** by Holte et al. (link)

Optimize Planning Heuristics to Rank, not to Estimate Cost-to-Goal by Leah Chrestien, Tomas Pevny, Stefan Edelkamp, Antonin Komenda, NeurIPS 2023 (link)

Bidirectional Heuristic Search

Abstract

We present MM, the first bidirectional heuristic search algorithm whose forward and backward searches are guaranteed to “meet in the middle”, i.e. never expand a node beyond the solution midpoint.

Key idea

MM runs an A*-like search in both directions, except that MM orders nodes on the Open list in a novel way. The priority of node n on $Open_F$, $pr_F(n)$, is defined to be:

$$pr_F(n) = \max(f_F(n), 2g_F(n)).$$