# Informed Search

CSCI 2400 "Part 3c"
Instructor: David Byrd

*Disclaimer:* Lecture notes can't and won't cover everything I say in class. You should attend class each day and use these for review or reinforcement.

# 6 Informed Search

We now have many different methods for choosing which discovered node to visit next. Instead of the first one, or the last one, or the one with the least cost from the start, why not just pick the *best* node? That is, the node *closest to the goal* that is also *along the correct path*? We could call it... *best-first search*.

## 6.1 Heuristics

Clearly there is a problem with our idea of always exploring the node closest to the goal along the correct path. To do that, we have to already know the final, correct path, and then why are we running search in the first place?

Okay, so we can't *know* that we are choosing the best node without already having a solution, but maybe we as problem designers do have some information that could serve as a *hint* or an *estimate* to the algorithm about which nodes are the most promising to explore. In other words, maybe we can inject some *domain knowledge* as additional information to the agent, allowing it to perform a kind of *Informed Search*.

If we considered a map of campus, with path/sidewalk intersections as nodes and path/sidewalk segments as edges, and we tasked a search agent with finding a path from Searles to Roux, it would have to use some brute force approach to exploring path segments. It lacks our "map intuition", our ability to quickly see that Roux is east-southeast of Searles and therefore we should probably take eastbound path segments. Somehow encoding that bit of knowledge for the agent is what creates an Informed vs Uninformed Search. We call that encoded domain knowledge, that hint or estimate, a *heuristic*.

Formally, a heuristic is simply a function that maps each state to a single nonnegative real number, which should represent an estimated cost to reach the goal from that state. In practice, it is outside knowledge about the specific problem that allows us to provide these estimates.

## 6.2 Best First Search

We can now describe *best-first search* or *BestFS*. It is the same general search algorithm as always, with an open priority queue, but now the priority is an estimated total cost of a path from the start to the goal through the node in question. Specifically, the priority is: $f(x) = g(x) + h(x)$, where $x$ is the state being placed in the open queue, $g(x)$ is the actual path cost incurred to reach $x$ from the start, and $h(x)$ is the value of the heuristic for state $x$ (i.e. an estimated remaining cost to the goal). Note that UCS can now be thought of as a specialization of BestFS such that $\forall x \in S : h(x) = 0$.

In general, BestFS will find a solution much faster than UCS, assuming our heuristic hint is helpful in some way. On the Romania map in the textbook, UCS expands a minimum of 12 nodes before dequeuing Bucharest, while BestFS with a straight-line distance heuristic expands only 5 nodes. Look what UCS has found after expanding three nodes:

$$(0, A), (75, AZ), (118, AT), (140, AS), (146, AZO), (229, ATL)$$

Compare BestFS at the same point:

$$(366, A), (393, AS), (447, AT), (449, AZ),$$
$$(413, ASR), (417, ASF), (415, ASRP), (526, ASRC)$$

UCS is trying to remain as close to the starting point as possible, while BestFS is exploring an estimate of the lowest *total* cost all the way from the start to the goal. It is no wonder BestFS is homing in much faster!

If you run the Romania example out to the end, you will find that BestFS discovers and enqueues a wrong (longer) path first, then finds the right (shorter) path *before it dequeues the wrong path*. This is why you must not halt search until you *dequeue* a goal state, because as in this example, halting when you *enqueue* a goal state is not guaranteed to find the best path.

If BestFS finds the same solution as UCS, but more quickly, could we do even better? What if we completely ignore the path cost so far, and just focus on getting closer to the goal? That is, use a priority of: $f(x) = h(x)$ only? This is *greedy best-first search*, and it will usually be the fastest search algorithm of all. However, while complete, it is *not optimal*. On our Romania map problem, it explores:

$$(366, A), (253, AS), (329, AT), (374, AZ), (178, ASF), (193, ASR), (0, ASFB)$$

and stops searching with a final path of $ASFB$ at a cost of 450, *which is worse than the optimal path* of $ASRPB$ with a cost of 428.

Best-first search is complete because it can search all states eventually. Its time and space costs are exponential, just like uniform cost search, though they can be quite decent with a good heuristic. Is it optimal? *No.* If the heuristic $h(x)$ *overestimates* the cost of the actual best path, the algorithm might never explore that path, instead stopping after discovering some other path that appears to have a lower cost (but does not).

Here is a short example demonstrating the non-optimality of BestFS when using an over-estimating heuristic:

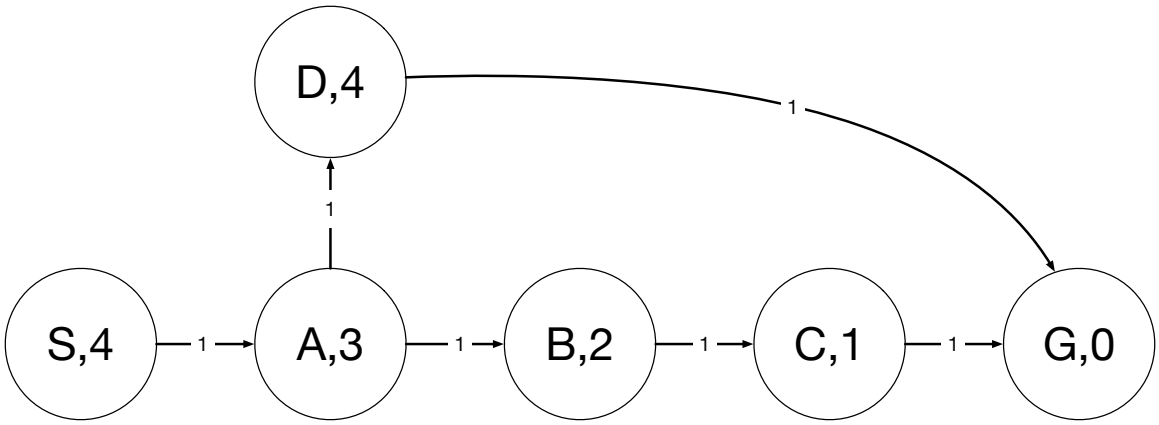


Figure 1: Non-optimality of Best First Search with over-estimating heuristic.

The heuristic given (next to the state name for each state) is perfectly reasonable except that it overestimates the cost from $D$ to the goal $G$. What happens when we run best-first search? At a certain point, the discovered states are:

$$(4, S), (4, SA), (4, SAB), (6, SAD), (4, SABC), (4, SABCG) \tag{1}$$

and the algorithm terminates with a best path of $SABCG$ at a cost of 4. In fact the best path was $SADG$ with a cost of 3, but the agent never explored this path due to the total cost estimate of 6 from the faulty heuristic.

But what if we could guarantee that our heuristic would *never* overestimate the cost to the goal? That is, the heuristic might optimistically make a potential path look *better* than it really is, but it would never make any path look *worse* than it is.

## 6.3 A* Search

*A* Search* is the name given to best-first search with an *admissible* and *consistent* heuristic. It turns out we already defined admissibility without realizing it. A heuristic is *admissible* if it never overestimates the actual cost from any state to the nearest goal. That is: $\forall x \in S : h(x) \leq h^*(x)$, where $h^*(x)$ is the optimal heuristic – one that gives the exact correct estimate for all states. Equivalently: $\forall x \in S : h(x) \leq c(x, G)$, where $c(x, G)$ is the minimum path cost from state $x$ to goal $G$. Was our straight-line distance heuristic in the Romania problem admissible? *Yes.* On a physical map it is not possible to reach one city from another traveling less than the straight-line distance between them.

A heuristic is *consistent* if it never overestimates the path cost between *any two states* in the graph. That is, the difference between the heuristic values for the two states must not exceed the actual cost to travel between them. Mathematically: $\forall x_1, x_2 \in S : \text{abs}\big(h(x_1) - h(x_2)\big) \leq c(x_1, x_2)$, where $c(x_1, x_2)$ is the minimum path cost from state $x_1$ to state $x_2$. Note that admissibility is simply a special case of consistency where $x_2 = G$. Thus, *consistency implies admissibility.*

For intuition on consistency, consider a specific case using a physical map problem. If $h(A) = 20$ and $h(B) = 10$, our heuristic estimates that city $A$ is 20 miles from the goal

$B$ is 10 miles from the goal. If it turns out that $c(A, B) = 5$, meaning we can travel from $A \to B$ in just 5 miles, there is a problem. Moving from $A$ to $B$, we have actually traveled 5 miles, but according to the heuristic we have gotten 10 miles closer to the goal! Either we have teleported, our world does not obey the expected laws of physics, or most likely, our heuristic is badly wrong. To rephrase consistency in terms of our example, if I walk 5 miles from $A$ to $B$, my estimated distance to the goal should *not* change by more than 5 miles!

With the admissibility requirement in place, our algorithm is guaranteed to return the optimal solution path. If the heuristic *underestimates* the cost to the goal, our agent may *waste time* exploring good-looking paths that turn out to be bad, but this cannot change its final answer. Wasting time exploring fruitless paths isn't good, so if we can also guarantee a heuristic that is *consistent*, then the algorithm will also be optimal in the sense of *exploring the fewest possible states* to find the minimum cost solution path.

You can find the original A* paper with a nice proof of optimality (in both senses) here: `https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf`

## 6.4 Comparing Heuristics

As we have seen, an admissible heuristic is sufficient to guarantee our search will return the correct solution path, and a consistent heuristic further guarantees we will explore no more states than necessary. In fact, it is quite hard to find a consistent heuristic for some problems and we will have to be content with just admissibility. In the case of two heuristics for the same problem, both of which are admissible but neither consistent, it would help to have some way of comparing them to see which is "better".

If two heuristics $h_1, h_2$ are both admissible, and $\forall x : h_1(x) \geq h_2(x)$, we say that $h_1$ *dominates* $h_2$. In other words, for every state, the $h_1$ estimate is a bigger number. But since we know $h_1$ never overestimates (it is admissible), this really says that for every state, the $h_1$ estimate is *closer to the correct value*. Since underestimating causes wasted exploration of wrong paths, a heuristic that underestimates *less* will use less computation to find the same answer.

What if neither heuristic dominates the other? If sometimes the $h_1$ estimate is better, but sometimes the $h_2$ estimate is better, we can instead simply *experiment*. We will run search many times using each heuristic and compare the results. To allow us to use variations of the problem, we normalize the result by the size of the state space, and call this the *informedness* of a heuristic.

Informedness $= \frac{\text{total size of state space}}{\text{avg \# states explored by h(x)}}$. The more *useful information* a heuristic contains, the higher its informedness will be, meaning it will find the correct answer while exploring a smaller portion of the total state space.

A final important consideration when selecting a heuristic is that it should be *fast to calculate*. Certainly it should be faster than running search on your state space. What do you gain by computing a heuristic that runs search and then running search, when you could have just run search in the first place?

*Acknowledgements and thanks to Professors Mark Riedl and Jim Rehg of the Georgia Tech School of Interactive Computing.*