Informed Search: A* Algorithm

Algorithmics, 186.814, VU 6.0

Günther Raidl

Algorithms and Complexity Group
Institute of Computer Graphics and Algorithms
TU Wien

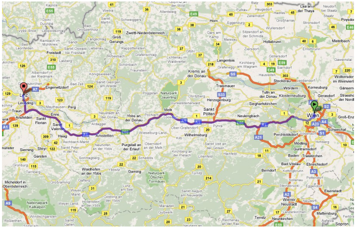
WS 2022/23, October 3, 2022

Literature for this Part

- S. Russel, P. Norvig: Artificial Intelligence – A Modern Approach, Algorithm Design, 3rd edition, Prentice Hall, 2009
- J. Pearl: Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, 1984

Path Planning – Ad Shortest Path Search

- Find a quickest/shortest route $s \rightsquigarrow t$ on a street network

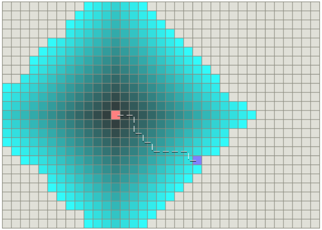


- Which shortest path algorithm to use?
- Only nonnegative weights $\rightarrow$ Dijkstra's Algorithm?
  - Simple implementation: $O(|V|^2)$
  - $\rightarrow$ Typically too slow for large street networks!
- How can we improve the situation?
  - Exploit properties of graph: sparse, approximately Euclidean
  - Apply heuristic information to speed up calculation!

Dijkstra's Algorithm: Uninformed Search

Dijkstra's algorithm always examines the closest not-yet-examined node.

Example: Simple grid graph



Here, Dijkstra's algorithm corresponds to Breadth-First-Search (BFS).

Informed Search:

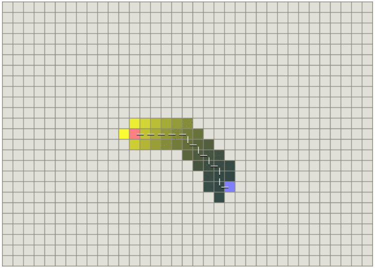
- Use an estimation/heuristic function to evaluate each already reached node
- Always expand a most promising node

Greedy Best First Search

Heuristic $h(x)$: "direct" (unobstructed) distance to goal t,

- e.g. Euclidean distance $d_2(x, t)$ or Manhattan distance $d_1(x, t)$

Always expand a reached node with minimum distance to goal

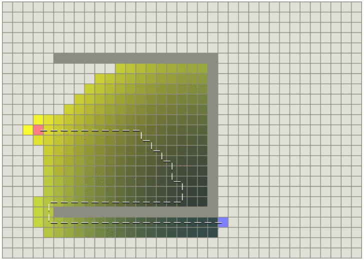


Here, the least number of nodes is expanded.

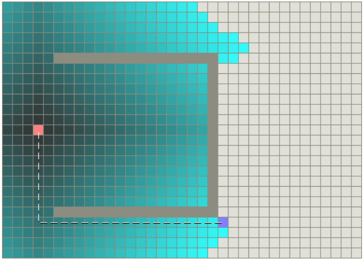
BUT: In general, does not guarantee optimal solution!

Greedy Best First Search May Fail

Greedy Best First Search:



Dijkstra's algorithm:



What can we do to avoid such stupid mistakes?
Can we combine the advantages to get a fast exact algorithm?

$\rightarrow$ A* algorithm

A* Algorithm – Basic Idea

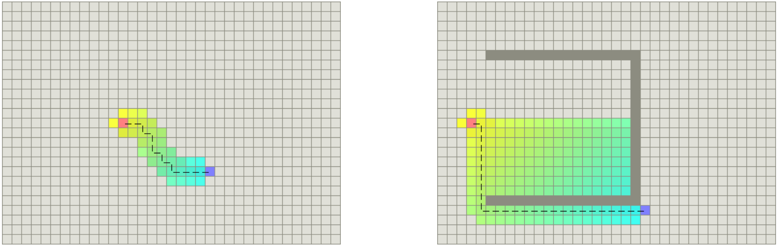
(Hart, Nilsson, Raphael, 1968)

Priority function: $f(x) = g(x) + h(x)$, $\forall x \in V$, where

$g(x)$: best cost so far to reach x

$h(x)$: estimated cost from x to goal t ("cost-to-go"), e.g. $d_1(x,t)$ or $d_2(x,t)$

I.e., $f(x)$ estimates the total cost from s to t via node x.

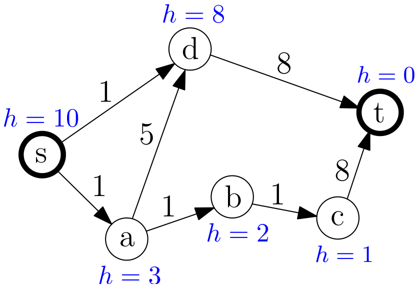


Implementation of A*

Algorithm 1: A* Algorithm

Input: digraph $G = (V, A)$ with $w_{ij} > 0\ \forall (i, j) \in A$, source node s, target node t;
$g(s) = 0$; $g(x) = \infty\ \forall x \in V \setminus \{s\}$;
priority queue `Q` $\leftarrow \{(s, f(s) = h(s))\}$;
while `Q` $\neq \emptyset$ **do**
    $x \leftarrow$ `Q.getMin()` // remove node with min. $f(x)$;
    if $x = t$ **then**
        return $f(t)$, path given by predecessor list pred(t);
    for all v *adjacent to* x **do**
        $g' \leftarrow g(x) + w_{xv}$;
        if $g' < g(v)$ **then**
            $g(v) \leftarrow g'$; pred(v) $\leftarrow x$ // new/better path to v;
            `Q`.put($v, g' + h(v)$)
return no path to target t;

Exemplary A* Run



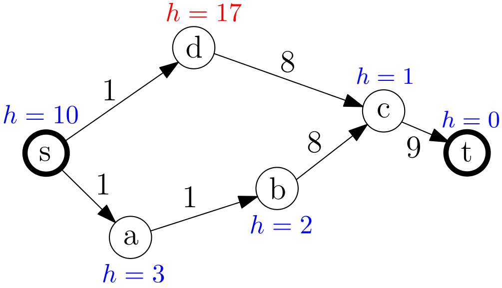
A* will first expand the nodes s, a, b, and c,
by which it has found the path (s, a, b, c, t) to the target.
It continues by expanding d,
finding the shortest path (s, d, t).

Important:
A* continues until the target node t is selected for expansion.

A*: Re-Expanding Nodes

A* may also find shorter paths to nodes that have already been expanded.

$\rightarrow$ Such nodes get re-expanded.



Now, s, a, b, and c are expanded, t reached; $f(c) = 10 + 1 = 11$. Then d is expanded, yielding a shorter path to c; $f(c) = 9 + 1 = 10$.

A^* Terminates

Theorem

A^* will always terminate in limited time.

Proof:

- The inner loop performs $\deg(x) = O(|V|)$ iterations, i.e. terminates.
- There are finitely many acyclic paths from s to other nodes in G.
- A^* only ever considers acyclic paths.
- On each major iteration a new acyclic path is considered because:
  - When a node is expanded the first time, a new path is considered.
  - When a node is re-expanded, a shorter and thus yet unconsidered route is considered.
- Thus, the most work A^* could do is to look at every acyclic path.

A^* Always Finds a Path if one Exists

Theorem

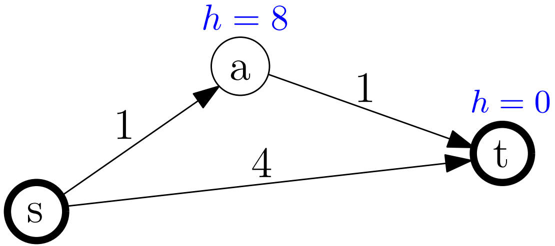
If a path from s to t exists, A^* will always find one.

Proof:

- Similarly as DFS and BFS:
- After expanding a node x, A^* has found paths from s to all nodes directly reachable via x.
- Only case when A^* stops without having found a path to t:
  - When no further reachable node exists that has not yet been expanded.
  - This, however, implies that no path to t exists.

Is A* Guaranteed to Find an Optimal Path?

No! At least not as introduced so far:



After expanding s, t will be selected for expansion and the algorithm stops with the path (s, t).

Under which conditions is A* guaranteed to find an optimal path?

Admissible Heuristics $h(x)$

Let $h^*(x)$ be the true minimal cost from x to t.

Definition (Admissible heuristic)

A heuristic $h(x)$ is admissible if it represents a lower bound, i.e.

$$h(x) \leq h^*(x) \qquad \forall x \in V.$$

- An admissible heuristic is "optimistic".
- In the grid-graph examples, $d_1(x, t)$ and $d_2(x, t)$ are admissible heuristics.
- $h(x) = 0$ also is an admissible heuristic.

A* with an Admissible Heuristic Yields an Optimal Path

Theorem

If $h(x)$ is admissible, A is guaranteed to find a least-cost path.*

Proof:

- Suppose it finds a suboptimal path with cost $f' > f^* = h^*(s)$.
- There must exist a node x which is
  - unexpanded
  - and whose path from s (as provided by `pred`) is the beginning of a true shortest path to t.
- $f(x) \geq f'$ (else A* would not have terminated), and

$$\begin{aligned} f(x) &= g(x) + h(x) \\ &= g^*(x) + h(x) \\ &\leq g^*(x) + h^*(x) \\ &= f^* \end{aligned}$$

- So, $f^* \geq f(x) \geq f'$, contradicting the assumption.

Monotonic Heuristics

Definition (Monotonic heuristic)

A heuristic $h(x)$ is monotonic (consistent) if

$$h(x) \leq w_{xy} + h(y) \quad \forall (x,y) \in A \qquad \text{and} \qquad h(t) = 0$$

- Monotonic heuristics ensure that for any path X from s to x

$$f(x) \;=\; g(x) + h(x) \;\leq\; g(x) + w_{xy} + h(y) \;=\; g(y) + h(y).$$

→ I.e., it is impossible to decrease $f(x)$ by extending a path to include a neighboring node.

→ No re-expansions of nodes.

- Comparable with the situation of non-negative edge weights in Dijkstra's algorithm.
- Monotonicity implies admissibility of the heuristic!

Monotonic Heuristics (contd.)

- Are $d_1(x,t)$ and $d_2(x,t)$ monotonic in our grid-graph examples?
  - Yes.

- For street networks finding a reasonable monotonic/admissible heuristic
  - is typically easier for finding shortest routes (e.g. Euclidean distance),
  - but not so for fastest routes.

Further Properties of A*

- A^* is optimally efficient w.r.t. the used heuristic $h(x)$, meaning that no complete algorithm employing the same heuristic will expand fewer nodes (proof omitted).

- Dijkstra's algorithm can be viewed as the special case of A* with $h(x) = 0$.

Runtime:

- in general $O(d^l)$, where l is the number of nodes on the shortest path, d a constant, but actual runtime depends strongly on $h(x)$

- better heuristic $\rightarrow$ better A* performance:
  if $h_1(x) \leq h_2(x) \; \forall x \in V \quad \rightarrow \quad h_2$ dominates (or is equal to) h_1
  $\rightarrow$ A* is guaranteed to expand no more nodes with h_2 than with h_1

- with an (almost) perfect heuristic $h(x) \approx h^*(x)$ runtime can be $\Theta(l)$

Example: 8-Puzzle

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

What heuristic to use?

- $h_0(x) = 0$
- $h_1(x)$=number of tiles in wrong positions
- $h_2(x)$=sum of Manhattan distances of each tile to its target location

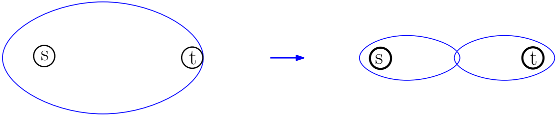
# moves, i.e., f^*	$\text{A}^*_{h_1}$-nodes	$\text{A}^*_{h_2}$-nodes	$\text{A}^*_{h_1}$-bf	$\text{A}^*_{h_2}$-bf
6	20	8	1,33	1.24
14	539	113	1,44	1,23
24	39.135	1.641	1,48	1,26

bf: branching factor, i.e., avg. number of exp. successors (Russel, Norvig; 2009)

Further Improvements/Variants

For complex problems, A* sometimes still too slow
and/or needs to much memory. Further improvement possibilities:

- Bidirectional search: search from s to t and t to s at the same time



- Weighted A*: "Inflate" heuristic, i.e. put more emphasis on $h(x)$
  - not admissible anymore, i.e. optimality not guaranteed
  - however, it might be possible to obtain a bound on sub-optimality (approximation guarantee)

- Preprocessing of data and store information for improved heuristic:
  - Extrem case: Calculate all-pairs shortest paths, store pairwise minimum distances, i.e. $h^*(x)$

Further Improvements/Variants (contd.)

- Hierarchical approach: store graph and compute shortest paths on a hierarchy of abstraction levels
  - E.g. street network:
    - First, plan rough route only considering major motor ways
    - Then refine, considering main roads connecting cities/villages
    - Finally, consider all smaller streets
- Iterated Deepening A* (IDA): primarily to limit memory usage
  - Consider integer costs; for costs $c = 0, 1, \ldots$ do
    - Perform depth-first search not expanding any node x with $f(x) > c$.
    - If t expanded, return path to it.
  - Guaranteed to find optimal solution but in general much slower than A*.