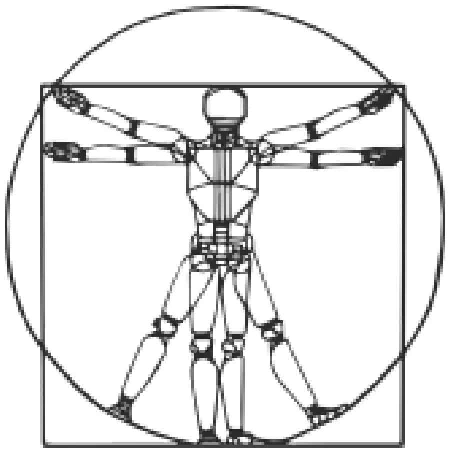
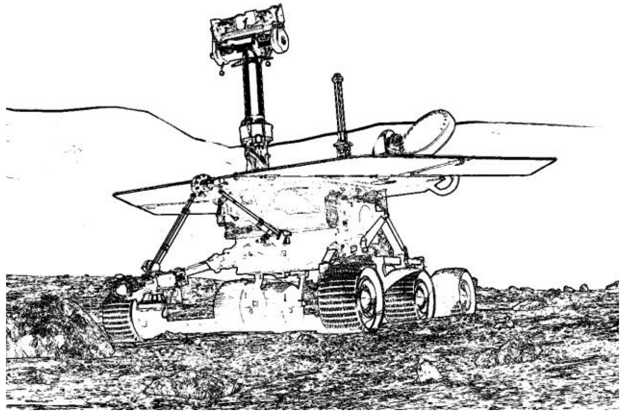




COMP417
Introduction to Robotics and Intelligent Systems

Lecture 8: Planning with Dijkstra and A*

Florian Shkurti

Computer Science Ph.D. student

florian@cim.mcgill.ca

McGill

MRL Mobile Robotics Lab at McGill University

Planning

- So far we have been trying to compute state-dependent **feedback controllers $u(x)=Kx$**

Planning

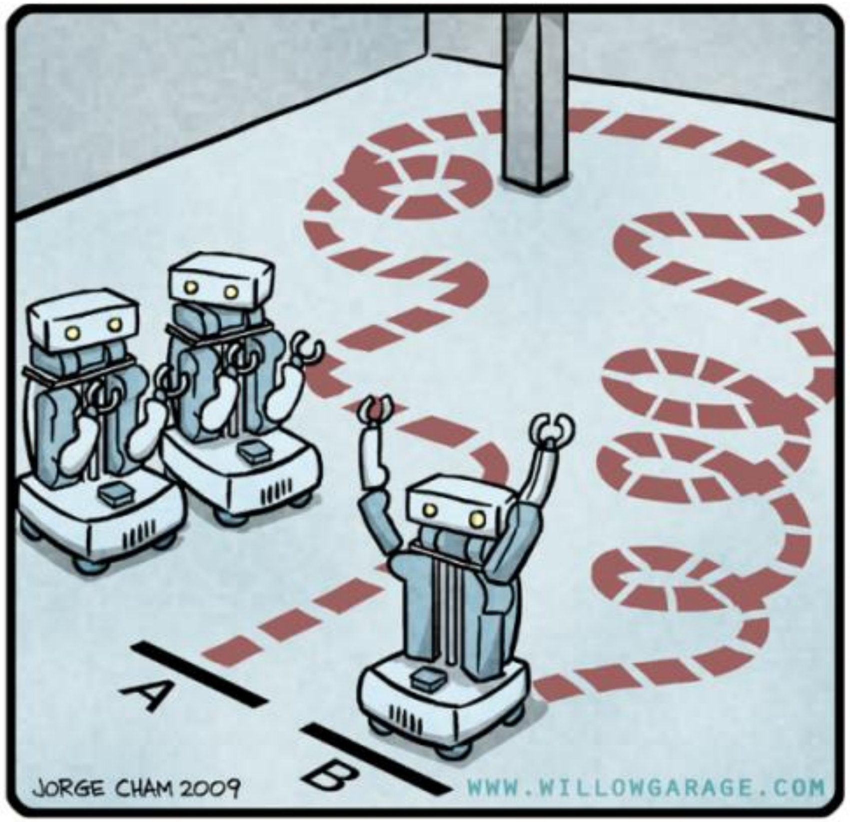
- So far we have been trying to compute state-dependent **feedback controllers u(x)=Kx**
- A plan is usually "open-loop," in the sense that it is assumed that once computed you can execute it perfectly
- This is not realistic because: wind, drag, external forces, friction, unknown factors make the system diverge from the planned trajectory.

Planning

- So far we have been trying to compute state-dependent **feedback controllers u(x)=Kx**
- A plan is usually “open-loop,” in the sense that it is assumed that once computed you can execute it perfectly
- This is not realistic because: wind, drag, external forces, friction, unknown factors make the system diverge from the planned trajectory.
- Planning does not usually take external disturbances into account. (External, independent feedback controllers have to make sure the robot is following the path closely)

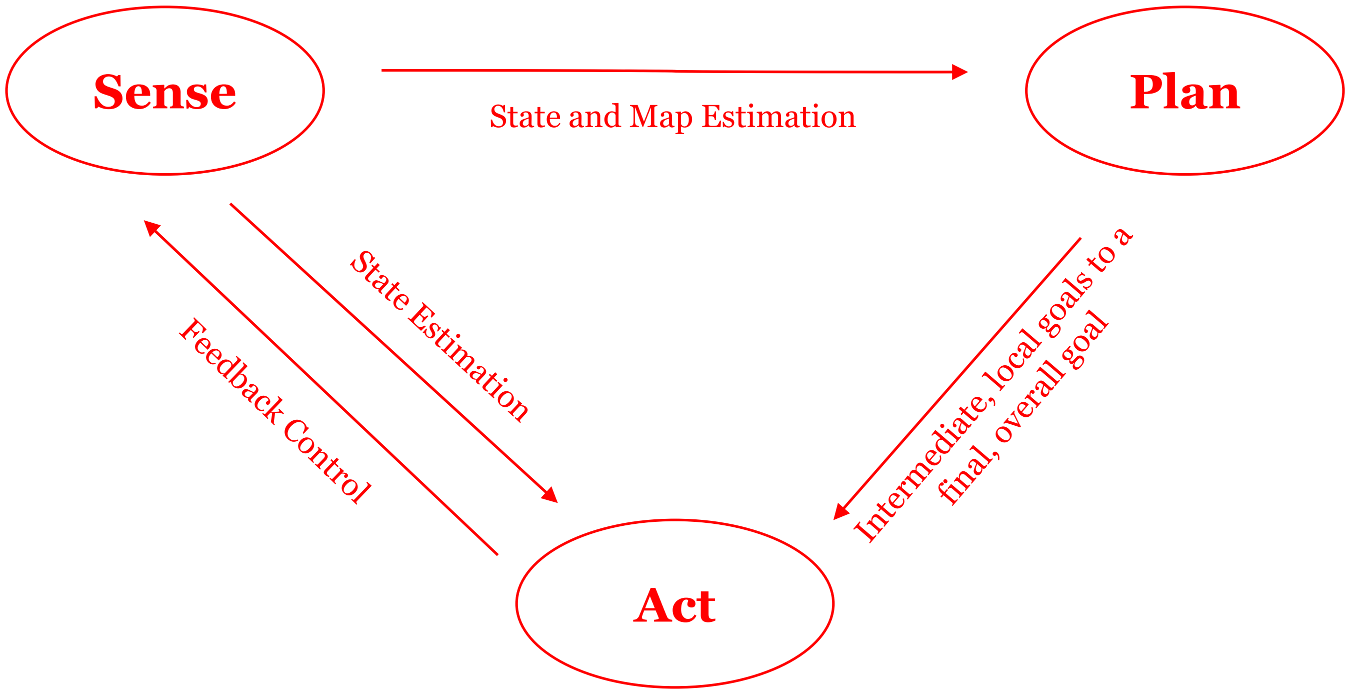
Why Bother Planning?

R.O.B.O.T. Comics

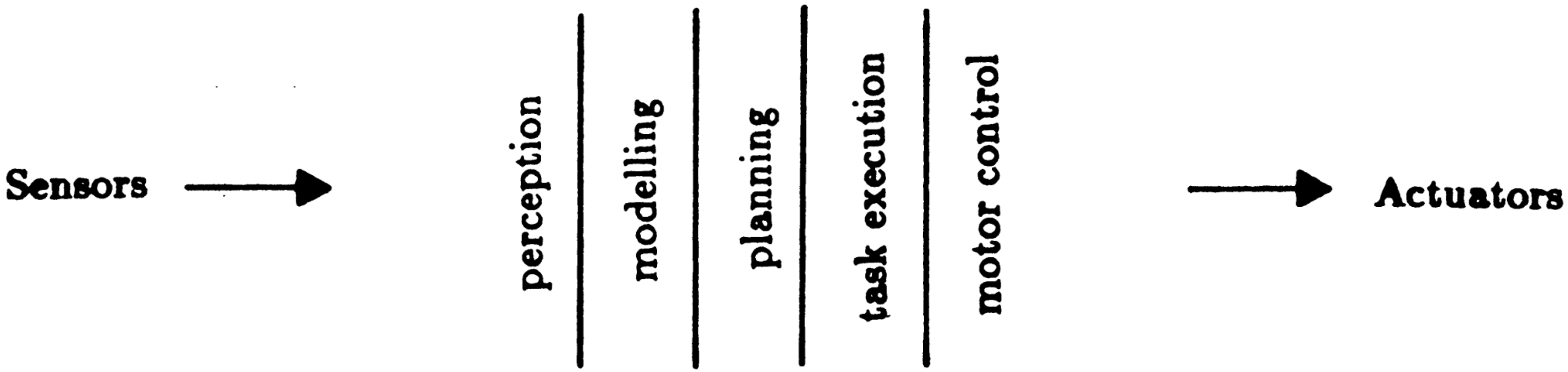


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Sense-Plan-Act Paradigm: Planning Is Necessary



Sense-Plan-Act Paradigm: Planning Is Necessary



Subsumption Architecture: Planning Is Not Necessary

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 303 September, 1987

PLANNING IS JUST A WAY OF AVOIDING FIGURING OUT WHAT TO DO NEXT

Rodney A. Brooks

Abstract. The idea of planning and plan execution is just an intuition based decomposition. There is no reason it *has to be* that way. Most likely in the long term, real empirical evidence from systems we know to be built that way (from designing them like that) will determine whether its a very good idea or not. Any particular planner is simply an abstraction barrier. Below that level we get a choice of whether to slot in another planner or to place a program which *does the right thing.* Why stop there? Maybe we can go up the hierarchy and eliminate the planners there too. To do this we must move from a state based way of reasoning to a process based way of acting.

Subsumption Architecture: Planning Is Not Necessary

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 303 September, 1987

PLANNING IS JUST A WAY OF AVOIDING FIGURING OUT WHAT TO DO NEXT

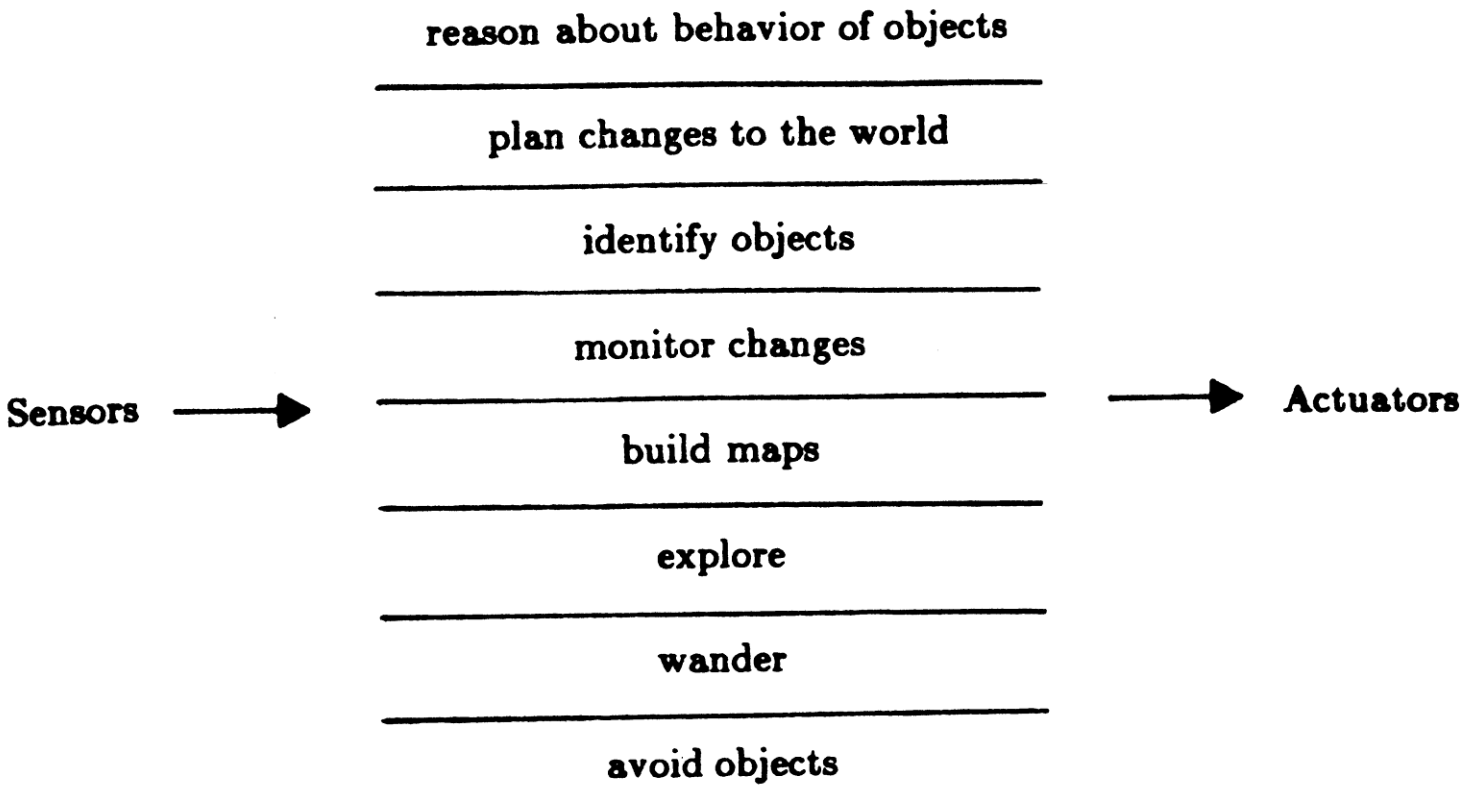
Rodney A. Brooks

Abstract. The idea of planning and plan execution is just an intuition based decomposition. There is no reason it *has to be* that way. Most likely in the long term, real empirical evidence from systems we know to be built that way (from designing them like that) will determine whether its a very good idea or not. Any particular planner is simply an abstraction barrier. Below that level we get a choice of whether to slot in another planner or to place a program which *does the right thing*. Why stop there? Maybe we can go up the hierarchy and eliminate the planners there too. To do this we must move from a state based way of reasoning to a process based way of acting.

He means: why bother estimating state and planning? It's too much work and could be error-prone. Why not only have a hierarchy of reactive behaviors/controllers?

One possibility:
instead of u(state)=…
use u(sensory observation)=…

Subsumption Architecture: Planning Is Not Necessary



Planning as graph search

- Graph nodes represent discrete states
- Edges represent transitions/actions
- Edges have weights

- Potential queries:
  - Shortest path from node a to node b, that does not hit obstacles
  - Is b reachable from a?

Planning as graph search

- Graph nodes represent discrete states
- Edges represent transitions/actions
- Edges have weights

- Potential queries:
  - Shortest path from node a to node b, that does not hit obstacles
  - Is b reachable from a?

- Typical assumptions:
  - Current state is known
  - Map is known
  - Map is mostly static

Dynamic Programming

$$D(v) = \min_{u \in N(v)} [d(v,u) + D(u)]$$

$$D(v_{\text{dest}}) = 0$$

Cost-to-go to destination
starting from node v

Instantaneous transition
cost needs to be non-negative

Dynamic Programming

$$D(v) = \min_{u \in N(v)} [d(v,u) + D(u)]$$

$$D(v_{\text{dest}}) = 0$$

Neighbors of v.
i.e. available actions

Cost-to-go to destination
starting from node v

Base case

Instantaneous transition
cost needs to be non-negative

Note: this should remind you
of the LQR cost-to-go update

$$J_{t+1}(\mathbf{x}) = \min_{\mathbf{u}} [g(\mathbf{x}_t, \mathbf{u}_t) + J_t(A\mathbf{x} + B\mathbf{u})]$$

$$J_0(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$$

Dynamic Programming

Worst-Case Complexity:

$O(|V|^2)$

In 2D grid world

$O(|V|)$

$$D(v) = \min_{u \in N(v)} [d(v,u) + D(u)]$$

$$D(v_{\text{dest}}) = 0$$

Cost-to-go to destination node starting from node v. Could also have denoted it

$D(v, v_{\text{dest}})$

Base case

Instantaneous transition cost for adjacent nodes needs to be non-negative

Note: this should remind you of the LQR cost-to-go update

$$J_{t+1}(\mathbf{x}) = \min_{\mathbf{u}} [g(\mathbf{x}_t, \mathbf{u}_t) + J_t(A\mathbf{x} + B\mathbf{u})]$$

$$J_0(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$$

Dijkstra’s algorithm

- Let $D(v)$ denote the length of the optimal path from the source node to node v (i.e. cost-to-come, not cost-to-go like before)
- Set $D(v) = \infty$ for all nodes except the source: $D(v_{\text{src}}) = 0$
- Add all nodes to priority queue Q with cost-to-come as priority
- While Q is not empty:

Dijkstra's algorithm

- Let $D(v)$ denote the length of the optimal path from the source node to node v (i.e. cost-to-come, not cost-to-go like before)
- Set $D(v) = \infty$ for all nodes except the source: $D(v_{\text{src}}) = 0$
- Add all nodes to priority queue Q with cost-to-come as priority
- While Q is not empty:
  - Extract the node v with minimum cost-to-come from the queue Q
  - If found goal then done
  - Remove v from the queue

  The cost-to-come of v is final at this point.
  Need to check if we can reduce the cost-to-come of its neighbors.

Dijkstra's algorithm

- Let $D(v)$ denote the length of the optimal path from the source node to node v (i.e. cost-to-come, not cost-to-go like before)
- Set $D(v) = \infty$ for all nodes except the source: $D(v_{\text{src}}) = 0$
- Add all nodes to priority queue Q with cost-to-come as priority
- While Q is not empty:
  - Extract the node v with minimum cost-to-come from the queue Q
  - If found goal then done
  - Remove v from the queue

    The cost-to-come of v is final at this point.
    Need to check if we can reduce the cost-to-come of its neighbors.
  - For u in neighborhood of v :
    - If $d(u, v) + D(v) < D(u)$ then
      - Update priority of u in Q to be $d(u, v) + D(v)$

Dijkstra's algorithm

- Let $D(v)$ denote the length of the optimal path from the source node to node v (i.e. cost-to-come, not cost-to-go like before)
- Set $D(v) = \infty$ for all nodes except the source: $D(v_{\text{src}}) = 0$
- Add all nodes to priority queue Q with cost-to-come as priority
- While Q is not empty:
  - Extract the node v with minimum cost-to-come from the queue Q
  - If found goal then done
  - Remove v from the queue

  The cost-to-come of v is final at this point.
  Need to check if we can reduce the cost-to-come of its neighbors.
  - For u in neighborhood of v :
    - If $d(u,v) + D(v) < D(u)$ then
      - Update priority of u in Q to be $d(u,v) + D(v)$

$O(1)$

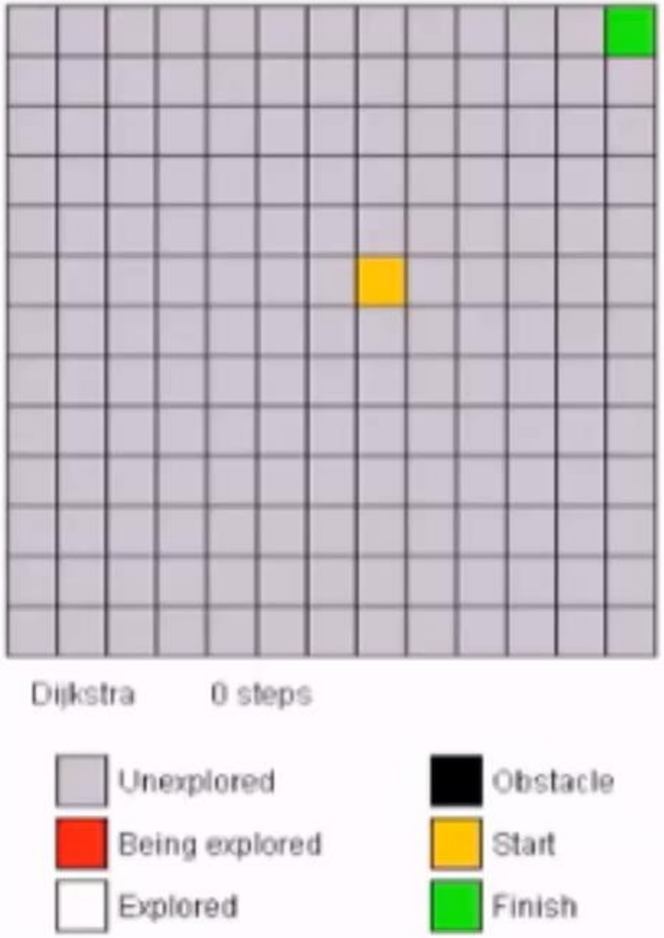
$O(\log|V|)$

$O(1)$

For Fibonacci heaps

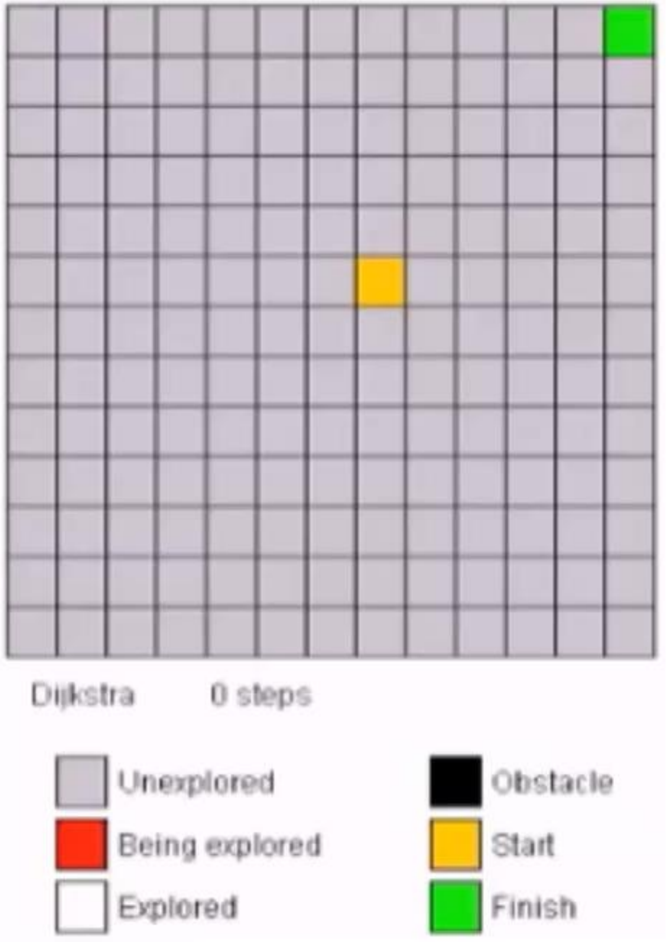
$$O(|E|T_{\text{update priority}} + |V|T_{\text{remove min}}) = O(|E| + |V|\log|V|)$$

Dijkstra’s algorithm: example runs



Dijkstra’s algorithm: example runs

Many nodes are explored unnecessarily. We are sure that they are not going to be part of the solution.



A* Search: Main Idea

- Modifies Dijkstra's algorithm to be more efficient
- Expands fewer nodes than Dijkstra's by using a heuristic

- While Dijkstra prioritizes nodes based on cost-to-come
- A* prioritizes them based on:

cost-to-come to v + lower bound on cost-to-go from v to v_{dest}

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

Lower bound on cost of path from source to destination that passes through v

Optimistic search: explore node with smallest f(v) next

A* Search: Main Idea

- Modifies Dijkstra's algorithm to be more efficient
- Expands fewer nodes than Dijkstra's by using a heuristic

- While Dijkstra prioritizes nodes based on cost-to-come
- A* prioritizes them based on:

cost-to-come to v + lower bound on cost-to-go from v to v_{dest}

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

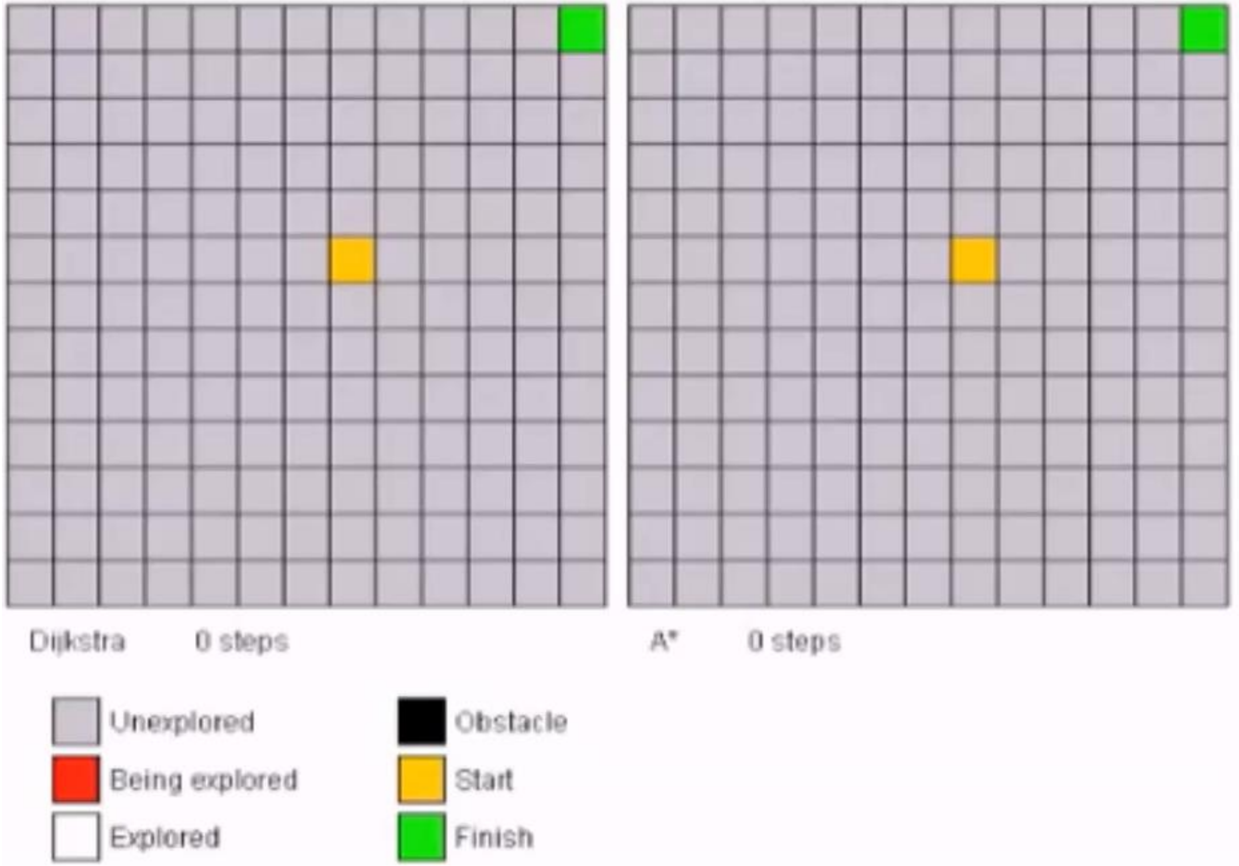
Lower bound on cost of path from source to destination that passes through v

h() is called a heuristic. h() must be **admissible,** i.e. underestimate the cost-to-go from v to destination. h() must also be **monotonic,** i.e. satisfy the triangle inequality.

A* Search

- Set $g(v) = \infty$ for all nodes except the source: $g(v_{\text{src}}) = 0$
- Set $f(v) = \infty$ for all nodes except the source: $f(v_{\text{src}}) = h(v_{\text{src}})$
- Add v_{src} to priority queue Q with priority $f(v_{\text{src}})$
- While Q is not empty:
  - Extract the node v with minimum $f(v)$ from the queue Q
  - If found goal then done. Follow the parent pointers from v to get the path.
  - Remove v from the queue Q
  - explored(v) = true
  - For u in neighborhood of v if not explored(u):
    - If u not in Q then
      - Add u in Q with cost-to-come $g(u) = g(v) + d(v, u)$ and priority $f(u) = g(u) + h(u)$
      - Set the parent of u to be v
    - Else if $g(v) + d(v, u) < g(u)$
      - Update the cost-to-come and the priority of u in Q
      - Set the parent of u to be v

Dijkstra vs A*



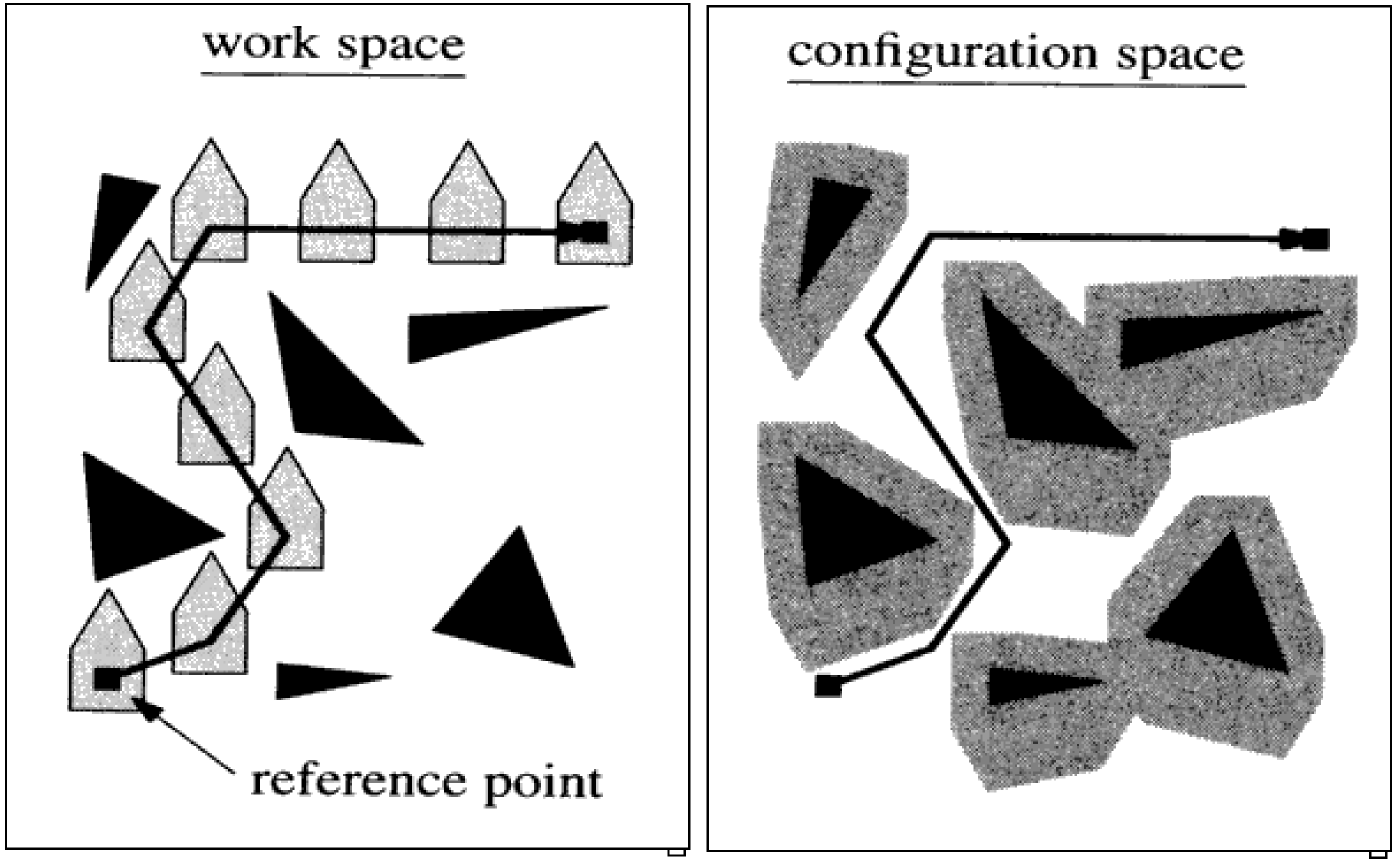
A* for cars

Configuration Space

Idea: dilate obstacles to account for the ways the robot can collide with them.

Why? Instead of planning in the work space and checking whether the robot's body collides with obstacles, plan in configuration space where you can treat the robot as a point because the obstacles are dilated.

This idea is typically not used for robots with high-dimensional states.



Configuration Space

How do we dilate obstacles?

Minkowski Sum

$$P \oplus Q = \{p + q \mid p \in P,\ q \in Q\}$$