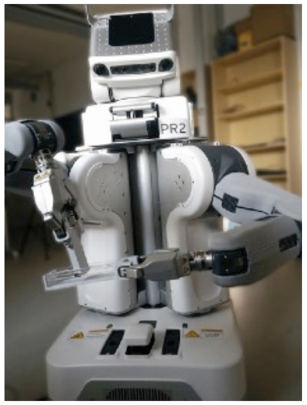
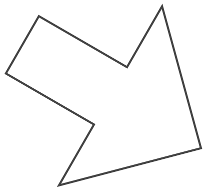
Last lecture ...

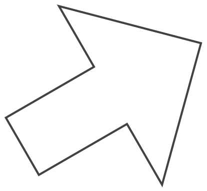
The configuration space enables us to represent robots of different shape, structure, … as a single point.

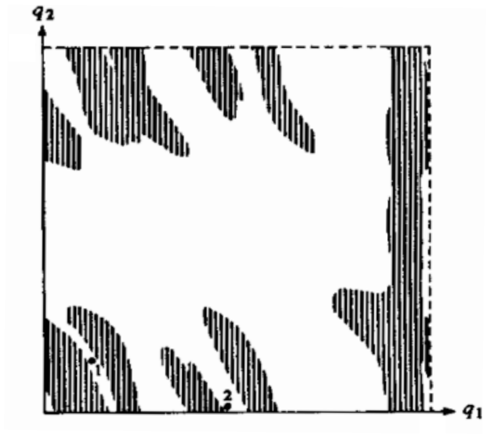
$q = (x, y, \theta)$



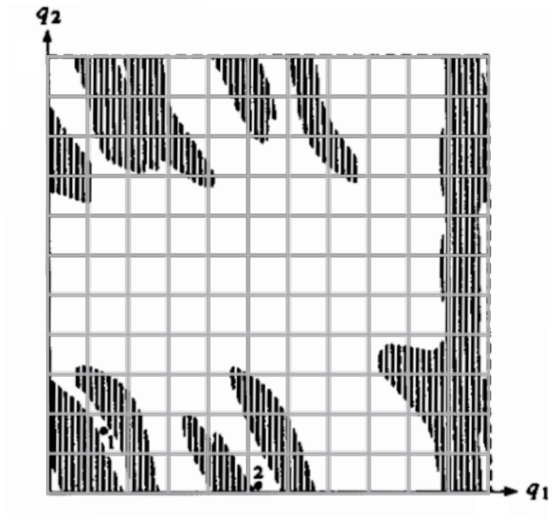
$q = (\theta_1, \theta_2, \ldots)$



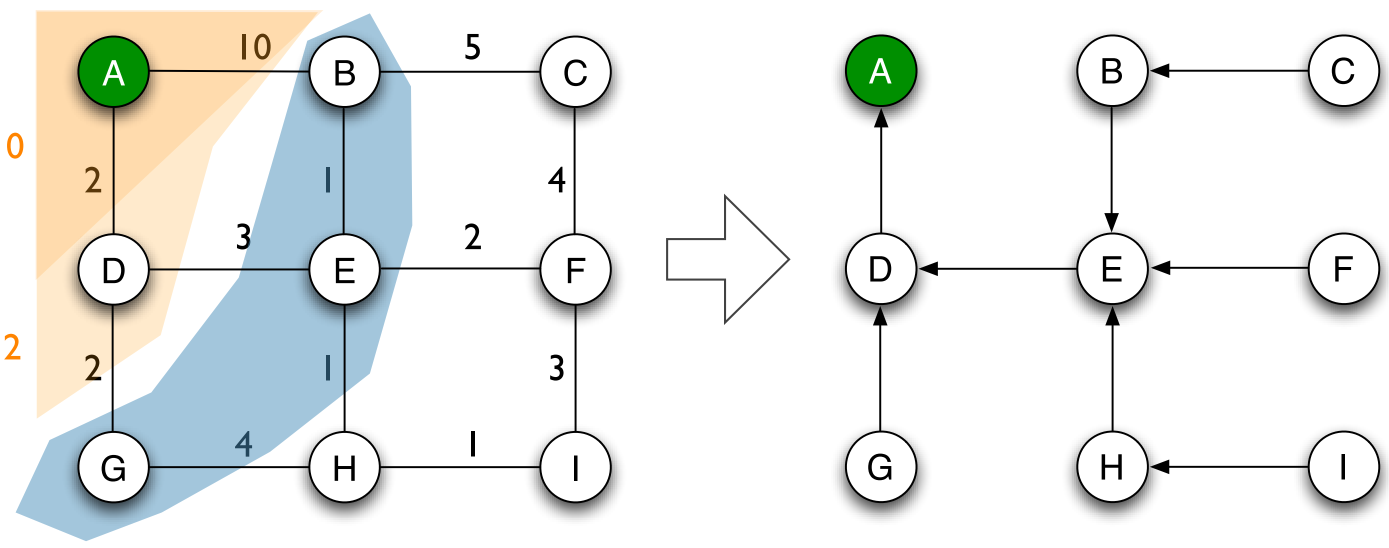




We place a grid of a suitable resolution on the C-space and create a cell-decomposition graph.



Finally, we search the graph, using a graph-search algorithm, e.g., the Dijkstra's algorithm for the shortest path.



Backward dynamic programming. Dynamic programming is a related, but different algorithmic approach to the shortest path problem. It is a very powerful, general idea for not only the shortest-path problem, but also a wide range of optimization problems. We use it for the shortest path here, but will encounter this idea several times later for other problems.

"Configuration" and "state" are often used interchangeably, though "state" is a more general notion. Here we follow the standard convention of DP algorithms and use "state".

Let $V(s)$ be the length of the shortest path to the goal for state s.

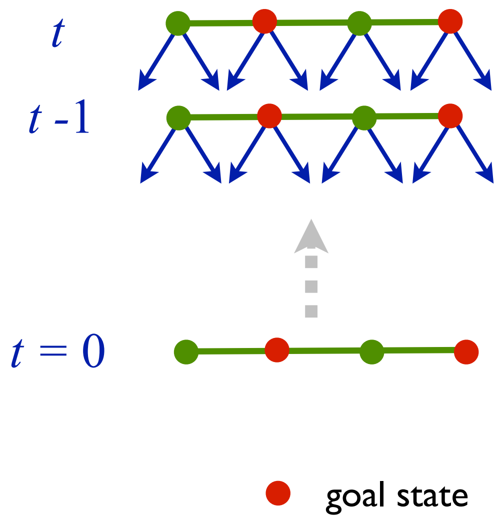
- Initialize
  - If s is a goal state, $V_0(s) = 0$
  - Otherwise $V_0(s) = +\infty$
- If there are t time steps to go, recurse

Node expansion
$$V_t(s) = \min_{s' \in A(s)} \left\{ \underbrace{c(s,s')}_{\text{the edge cost from } s \text{ to } s'.} + V_{t-1}(s') \right\}$$

nodes adjacent to s

for $t = 1, 2, 3, \ldots, N$-1, where $A(s)$ is the set of nodes adjacent to s, including s.

It is sufficient to choose N to be the total number of nodes, as a shortest path visits each node at most once.



Dynamic programming finds the shortest path from every state to a goal.

Both the computation time and the storage space required depend on the size of the state space, i.e., the **total number of states**.

Question. Would dynamic programming work well for

- a rigid body translating and rotating in the plane?
- a hyper-redundant robot arm consisting of 10 rigid pieces linked together at the joints?

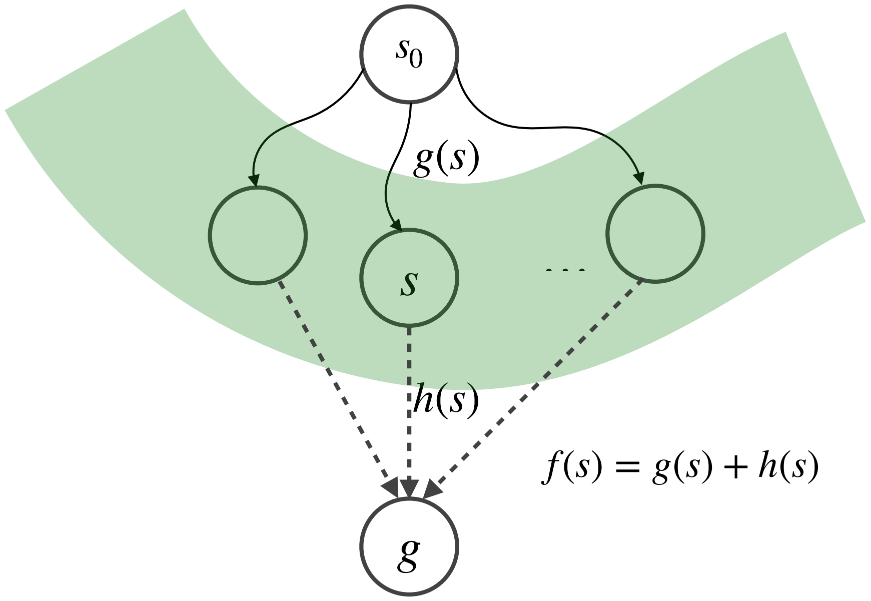
The number of states grow exponentially with the dimension of the C-space (number of DoFs).

Forward search. Dynamic programming is not practical for very large state space. In practice, we often care about the shortest path from the current state s_0.

The forward search algorithm starts at the current state and performs lookahead search until it reaches a goal state.

- Start at the current state s_0.
- Maintain a priority queue Q. Insert s_0 into Q.
- Choose a node s from Q, according to the priority $f(s) = g(s) + h(s)$. For each node s' incident to s, insert s' into Q with updated value of $f(s)$. Repeat until reaching a goal node g.

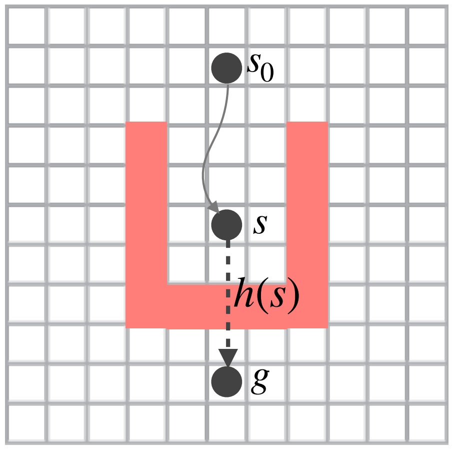
Node expansion



Intuitively, $g(s)$ is the cost-to-come (from s_0 to s) and $h(s)$ is the cost-to-go (from s to g).

$g(s)$ is the length of the shortest path found so far to go from s_0 to s. $h(s)$ is a heuristic estimating the shortest path from s to g. $g(s)$ represents the "true" shortest path information, found through search; $h(s)$ represents our guess.

- Uninformed search: $f(s) = g(s)$
  Uninformed search sets $h(s) = 0$, effectively ignoring the heuristic. It finds the shortest path. However, the search does not consider the goal and is thus inefficient. An example is Dijkstra's algorithm.
- Greedy search: $f(s) = h(s)$
  Greedy search ignores g(s), effectively not remembering anything about the past search and only trusting the heuristic estimate $h(s)$. However, the heuristic is sometimes misleading. Greedy search may get stuck in a local minimum and fail to find a shortest path.
- A* search: $f(s) = g(s) + h(s)$
  A* search combines information from "cost-to-come" $g(s)$ and "cost-to-go" $h(s)$. It finds the shortest path if the heuristic function $h(s)$ is **admissible**.



Admissibility. A heuristic function $h(s)$ is admissible if it provides an optimistic estimate, in other words, an underestimate of the true shortest path length from s to the goal.

Efficiency. Good heuristics significantly improve computational efficiency of search algorithms. Consider the following two extremes. One extreme is an uninformed heuristic $h(s) = 0$. It is clearly admissible, but not very useful. Suppose that we use it to search a grid graph of 1000x1000, with the lower-left corner and the upper-right corner, as the start and the goal, respectively. With $h(s) = 0$, the A* algorithm expands all1,000,000 nodes, just like the Dijkstra's algorithm. The other extreme is a fully informed heuristic. With the fully informed heuristic, the A* algorithm expands only the nodes on the shortest path, i.e., only about 2,000 nodes in total.

Question. What is the fully informed heuristic?

The computation time of forward search does not depend on the total number of states directly. What does it depend on then? The computation time of a tree search algorithm depends on the **branch factor** and the **depth** of the search tree.

The fully informed heuristic is the shortest path length itself Although we don't know it in advance, we can try to come up with approximations.

Heuristics are critical to the performance of forward search algorithms. Here are some general ideas for obtaining effective heuristics:

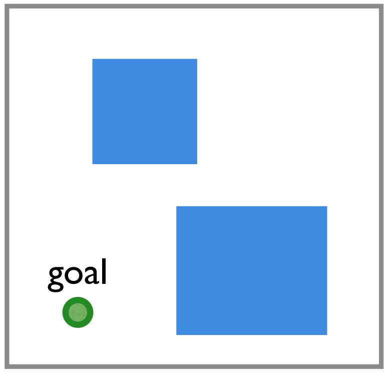
- Use domain-specific knowledge.
- Relax motion constraints the path and calculate the cost of the relaxed path.
- Learn heuristics from experiences.
    - Ask ChatGPT (2023 AD) 😊

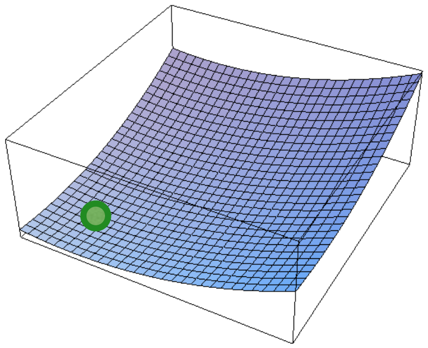
A common heuristic function for robot path planning is the Euclidean distance, i.e., the length of the straight-line path to the goal.

Question. Is this heuristic admissible?

It is admissible because it removes the obstacle constraints.

Artificial potential field. In essence, the well-known potential field method for motion planning is simply a handcrafted heuristic function based on *a priori* domain knowledge.

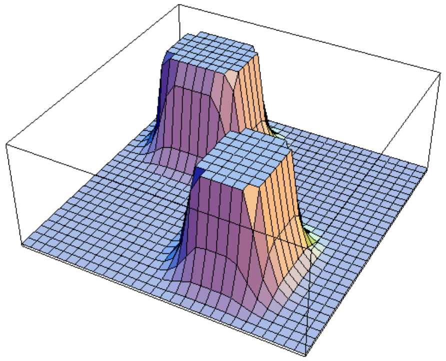




Attractive potential

Activate the attractive potential if the distance to the goal is smaller than a chosen value P.

$$U_{\text{att}}(q) = \frac{1}{2}\xi\left(d(q, q_{\text{goal}})\right)^2 \quad \text{if } d(q, q_{\text{goal}}) \leq P$$



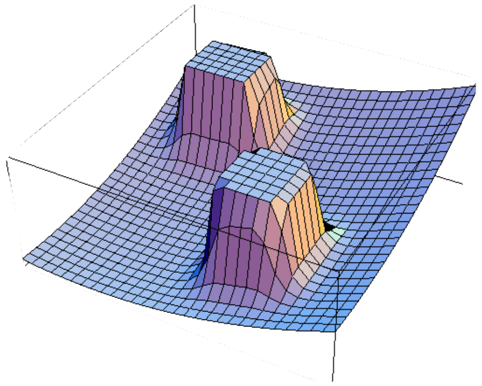
Repulsive potential

Activate the repulsive potential if the distance to an obstacle is smaller than a chosen value Q.

For each obstacle,

$$U_{\text{rep}}(q) = \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q}\right) \quad \text{if } D(q) \leq Q$$

distance from q to the obstacle



$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

Choose an action to follow the gradient of the artificial potential field $\nabla U = \nabla U_{\text{att}}(q) + \nabla U_{\text{rep}}(q)$.

Question. As a heuristic, the artificial potential field is intuitively appealing. How does it fail?

Learn a heuristic. The greedy algorithm finds an optimal path if it uses the fully informed heuristic, i.e., the true optimal path length. Of course, the fully informed heuristic is not known in advance, but maybe we can **learn** to approximate it.

From the robot's current state s, it chooses the next s' according the estimated cost-to-go:

node expansion

$$f(s) = \min_{s' \in A(s)} \{c(s,s') + h(s')\}$$

In the standard greedy search, the heuristic h is chosen in advance and does not change during the search. To learn, we set

$$h(s) = f(s)$$

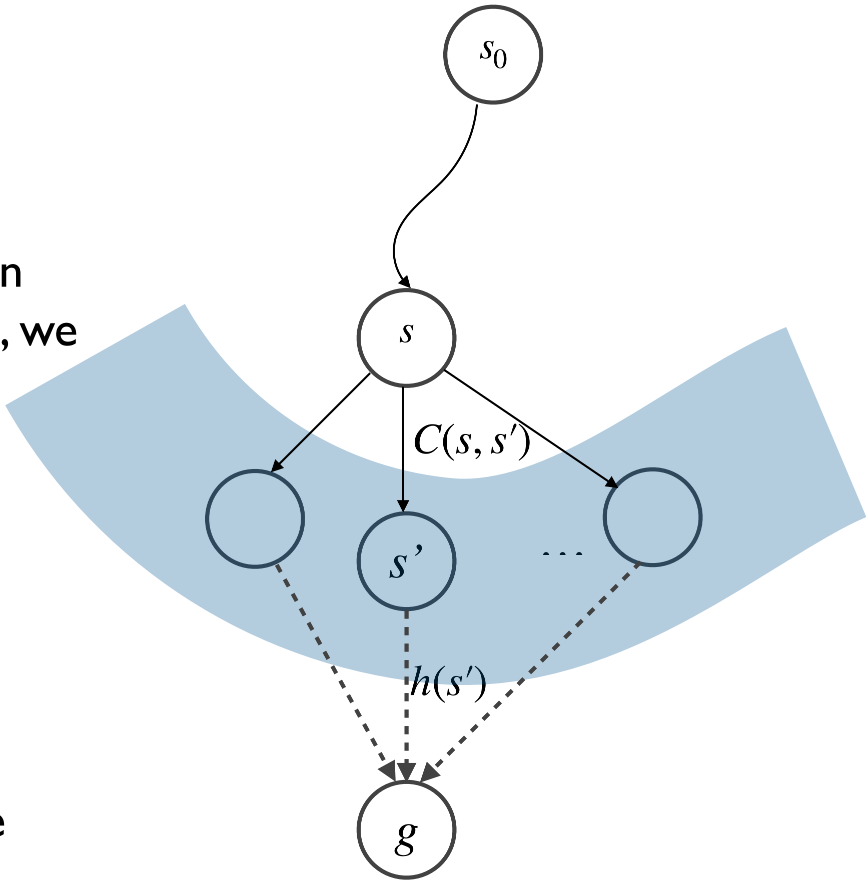
in each step to get an improved heuristic.

Putting the two steps together, we have

$$h(s) \leftarrow \min_{s' \in A(s)} \{c(s,s') + h(s')\}$$

Why does this improve the heuristic? Compare with the recurrence for dynamic programming

$$V_t(s) = \min_{s' \in A(s)} \left\{c(s,s') + V_{t-1}(s')\right\}$$



So heuristic improvement is simply one step of dynamic programming at a particular state s. If we repeat this enough number of steps over all states, then $h(s) = V^*(s)$, i.e., the true shortest path length, the perfect heuristic.

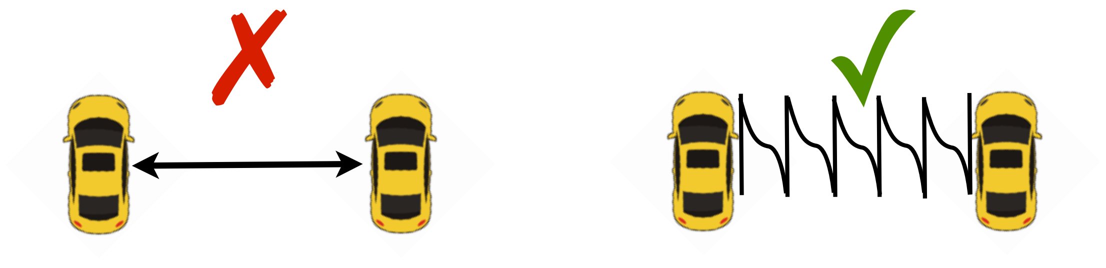
This is a simple example of **learning** by memorizing past experiences.

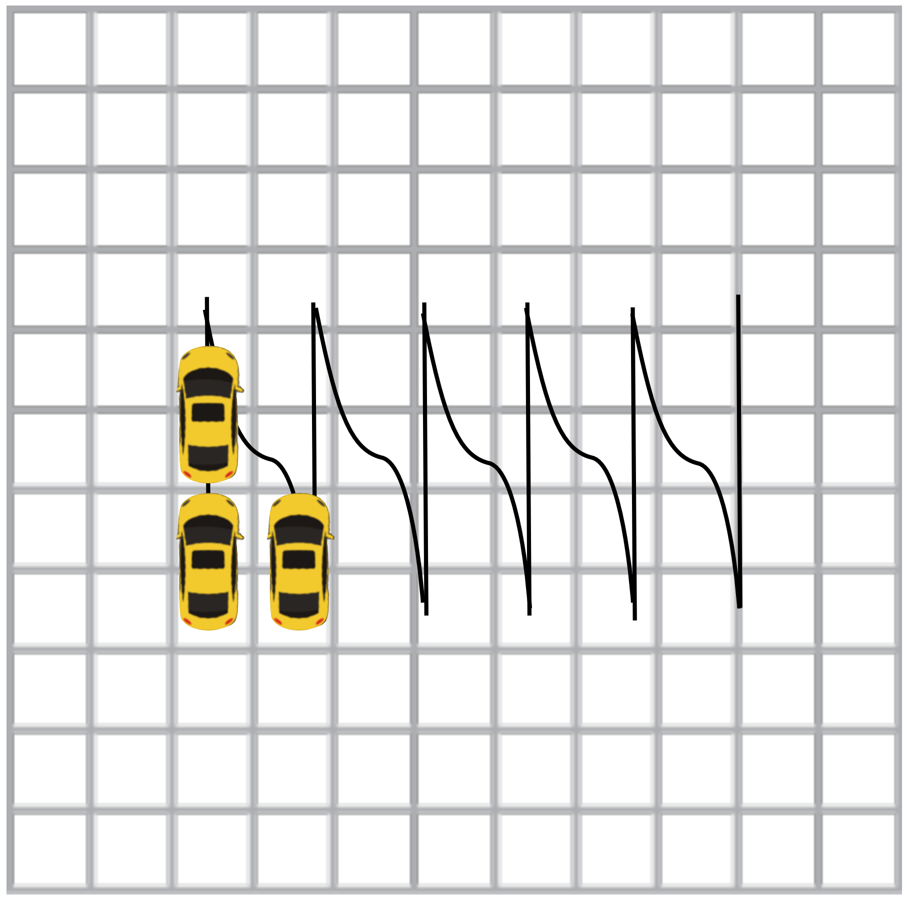
Our derivation also shows that we can perform t steps of dynamic programming for any t and use the resulting $V_t(s)$ as a heuristic function.

This is the main idea of learning real-time A (LRTA*).*
For more details, see

H. Geffner and B. Bonet. Solving large POMDPs using real time dynamic programming. In Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes, 1998.

Question. Can we use the cell-decomposition graph to plan the motion of a car?





Not directly. The adjacent free cells may not be directly traversable because of motion constraints.

Constraints on motion. Robot motion is subject to various constraints.

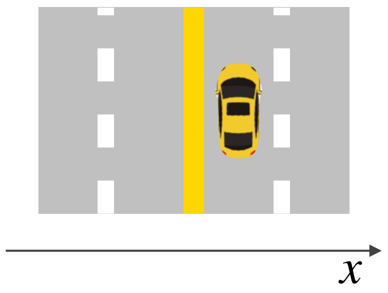
A constraint is holonomic if it involves only the configuration q:

$g(q) = 0$ or $g(q) \leq 0$

A holonomic constraint restricts the set of valid configurations.

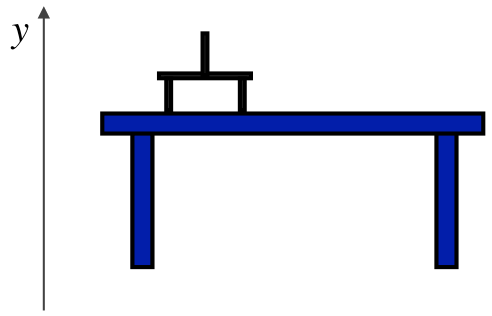
Example. The in-lane driving constraint is holonomic.

$0 \leq x \leq 1$



Example. The contact constraint is holonomic. The constraint $y = 0$ ensures that the robot gripper is in contact with the tabletop.

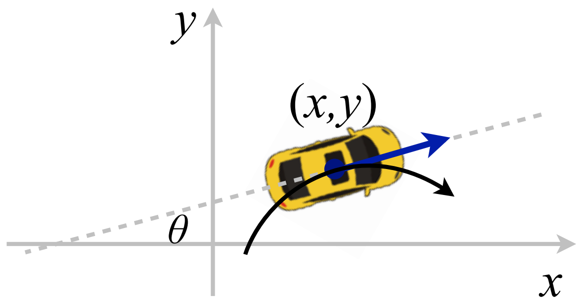
$y = 0$



A constraint is nonholonomic if it involves both configuration q and configuration velocity $\dot{q}$:

$$g(q, \dot{q}) = 0 \text{ or } g(q, \dot{q}) \leq 0$$

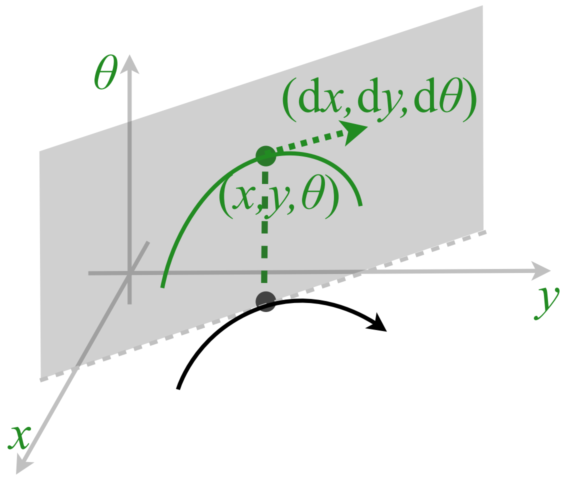
A nonholonomic constraint restricts the set of valid configuration velocities, but not necessarily the set of configurations.



Example. A car cannot drive sidewise. The instantaneous velocity of the car must align with the current orientation of the car:

$$\tan\theta = \frac{\dot{y}}{\dot{x}}$$

$$\Rightarrow \dot{x}\sin\theta - \dot{y}\cos\theta = 0$$



The constraint involves both the configuration $q = (x, y, \theta)$ and the configuration velocity $\dot{q} = (\dot{x}, \dot{y}, \dot{\theta})$. This constraint restricts the instantaneous velocity of the car. However, it does not restrict the set of reachable configurations: a car still can reach any arbitrary configuration under the constraint. Otherwise, the car would not be very useful!

Under the above constraint, we can write down the system equation as

speed v

orientation θ

$$
\begin{aligned}
\dot{x} &= v \cos\theta \\
\dot{y} &= v \sin\theta \\
\dot{\theta} &= (v/L)\tan\phi
\end{aligned}
$$

steering angle ϕ

distance L between front and rear wheel axes

Which indeed satisfied the nonholonomic constraint. It is a controlled dynamic system of the form

$$\dot{q} = f(q, u),$$

where control u consists of the speed v and steering angle ϕ.

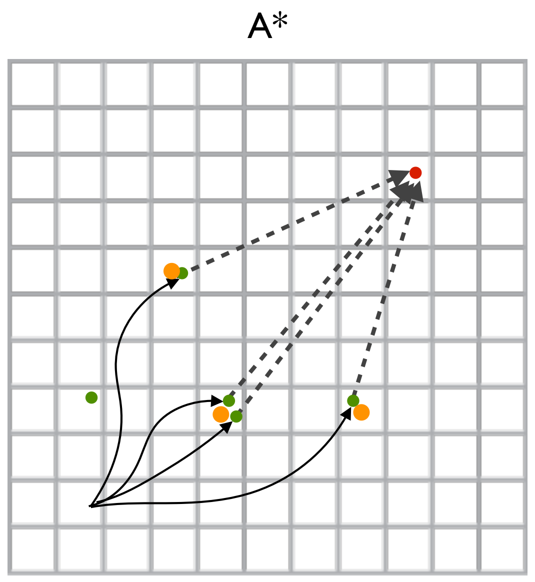
Suppose that we apply some control u at configuration q for a fixed time duration Δt. We arrive at a new configuration

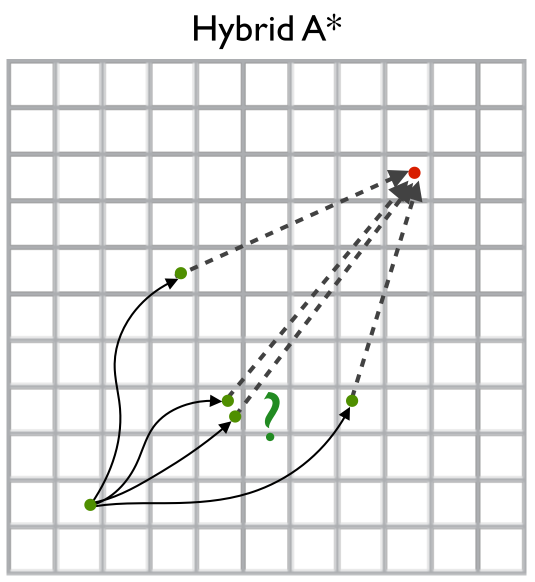
$$q_{t+1} = q_t + \dot{q}_t \Delta t,$$

thus allowing us to "connect" q_t to q_{t+1} via the control u.

Hybrid A*. The standard A* algorithm associates with each node a configuration at the center of a grid cell. The hybrid A* algorithm associates with each node a **continuous** configuration $q = (x, y, \theta)$.

Question. Is it a good idea to associates with each node a configuration at the center of a grid cell? Why must we use a continuous configuration?

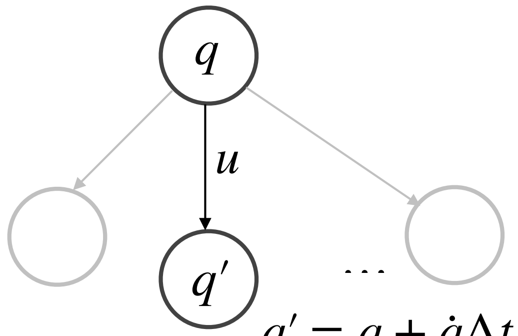




Even though two configurations within a cell are close in Euclidean distance, they may be far apart in terms the shortest path that connects them because of the motion constraint. Two car poses side by side are close in Euclidean distance, but the car cannot move sidewise!

If the search reaches a node visited earlier, we can retain only one associated continuous configuration, the one with lower value of $f(q)$.

We now apply the hybrid A* algorithm to car driving. To expand a node,



$q' = q + \dot{q}\Delta t$, where $\dot{q} = f(q, u)$,

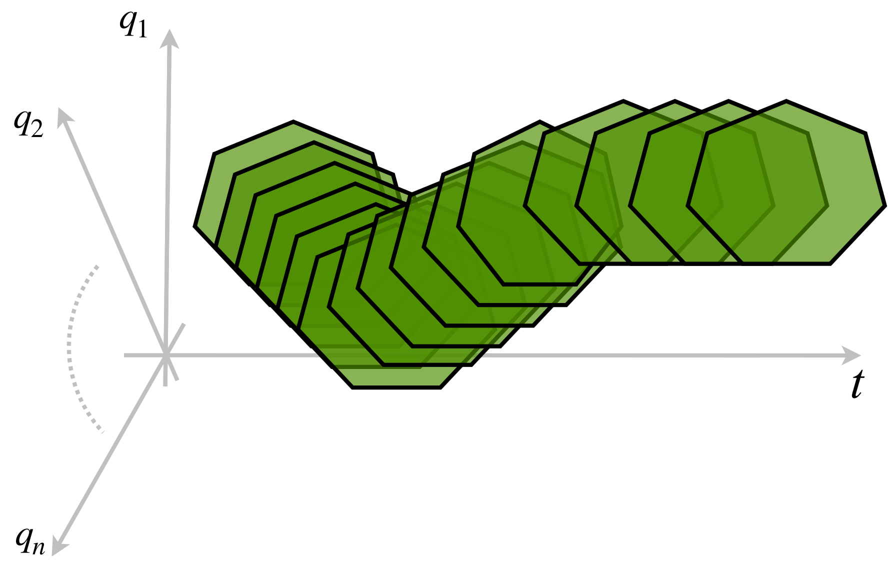
To use the hybrid A* algorithm effectively, we need a good heuristic.

- Nonholomic without obstacles $h_1(q)$. The heuristic calculates the shortest nonholomic path to the goal, ignoring the obstacles.
- Holonmic with obstacles $h_2(q)$. The heuristic calculates the shortest collision-free path to the goal, ignoring the nonholonmic constraints.

Question. Show that both $h_1(q)$ and $h_2(q)$ are admissible.

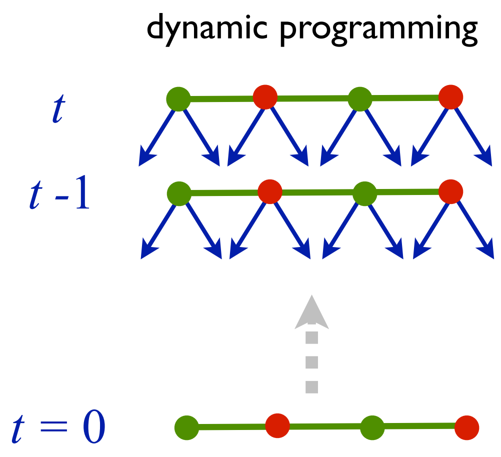
Question. Show that $h(q) = \max\{h_1(q), h_2(q)\}$ is a better heuristic than either $h_1(q)$ or $h_2(q)$.

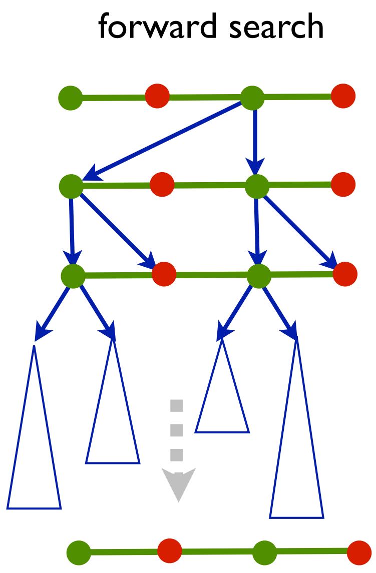
Dynamic environments. So far, we assume that the environment is known and does not change over time. This is somewhat true in a tightly controlled factory. More often, the environment changes. Consider an autonomous vehicle faced with many moving pedestrians. As a first step towards this more realistic setting for motion planning, we assume that future pedestrians are known or predictable. In this case, we just need to add a time axis to the configuration space to form the configuration space-time. The same algorithms that we have studied so far then all apply without change.



Summary.

- Dynamic programming finds the shortest path for every state. The computation time depends on the size of the state space.
- Forward search finds the shortest path for the current state. The computation time depends on the branch factor and the depth of the search tree.
- The efficiency of forward search depends on the choice of the heuristic function.
- A holonomic constraint involves the configuration only. It restricts the set of valid configurations. A nonlonomic constraint involves both the configuration and the configuration velocity. It restricts the set of valid configuration velocities, but not necessarily the set of configurations.
- Hybrid A* solves the shortest path problem with continuous states.
- We can use hybrid A* to solve for motion planning of a car in a dynamic environment if we know or can predict the future reliably. However, such prediction is not always possible.





Required readings.

Supplementary readings.

- For details on the hybrid A* algorithm, read the relevant section in the paper:
  Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robotics Research*, 29(5):485–501, 2010.
  Or if you prefer very detailed explanations, watch this video.
- For details on learning the heuristic, read
  H. Geffner and B. Bonet. Solving large POMDPs using real time dynamic programming. In Working Notes of *AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 61–68, 1998.
- For background on heuristic search, read the relevant chapters in the classic textbook
  Artificial Intelligence: A Modern Approach by S. Russel and P. Norvig.

Key concepts.

- Dynamic programming
- Forward search
- Holonmic and nonholomic constraints
- Hybrid A* algorithm