Motion Planning Lecture 3

Graph-based Planning: Representations, A*, Admissible heuristics

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

May 8, 2024

Recap Last Week

- How to model configuration spaces of arbitrary robots: Topological spaces
- How to measure distances: Metric spaces
- How to make your robot do the right thing: Constraints and collision checking

Today

- Building graphs: Find representations of configuration space
- A* algorithm: Optimal paths over graphs and optimality proof
- Admissible heuristics: How to better inform graph search

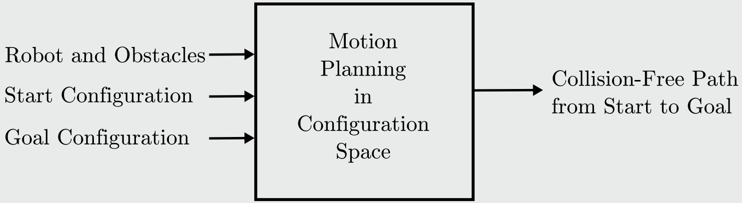
Graph-based planning: Motivation

(Geometric) Motion Planning Problem

Main Idea

Any robot can be modelled as a point in a configuration space (Paper by Lozano-Pérez and Wesley [1] (1979)).

The Motion Planning Problem



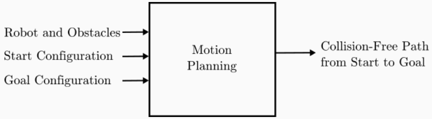
(Geometric) Motion Planning Problem

Requirements

- A configuration space $\mathcal{Q}$
- Constraints to distinguish $\mathcal{Q}_{\text{free}}$ and $\mathcal{Q}_{\text{obs}}$
- An initial configuration $\mathbf{q}_{start} \in \mathcal{Q}_{\text{free}}$
- A goal configuration $\mathbf{q}_{goal} \in \mathcal{Q}_{\text{free}}$

Outcome

- A collision free sequence $\mathbf{q} : [0, 1] \to \mathcal{Q}_{\text{free}}$ such that $\mathbf{q}(0) = \mathbf{q}_{start}$ and $\mathbf{q}(1) = \mathbf{q}_{goal}$.
- *Complete* algorithm: Find a sequence $\mathbf{q}(\cdot)$ if one exists, or report that no such path exists.



Solving (Geometric) Motion Planning Problems

Standard two-step approach:

- (1) Find a representation of the configuration space
- (2) Use representation to compute a path

Solving (Geometric) Motion Planning Problems

Mapping-based approach.

Cover-based approach.

Graph-based approach.

Solving (Geometric) Motion Planning Problems

Mapping-based approach.

- (1) Map all obstacles into the configuration space and decompose space into cells
- (2) Find an optimal solution by connecting cells

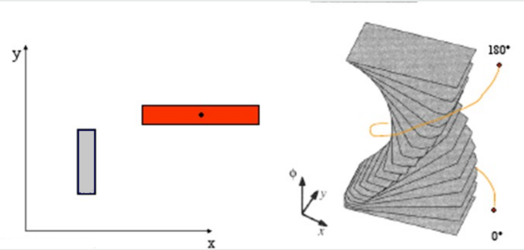
Robot Motion Planning (1991) by Jean-Claude Latombe

Cover-based approach.

Graph-based approach.

Solving (Geometric) Motion Planning Problems

Mapping-based approach.



Left: Robot (red) and obstacle (grey). Right: Configuration space; each polygon is Minkowski sum of robot and obstacle at fixed orientation.

Cover-based approach.

Graph-based approach.

Solving (Geometric) Motion Planning Problems

Mapping-based approach.

Cover-based approach.

- (1) Find *open sets* covering the configuration space
- (2) Find an optimal solution by connecting open sets

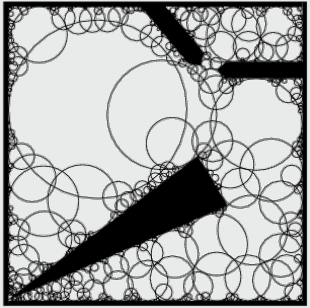
Computing a Composition of Funnels (LaValle, Planning Algorithms, 2006) `http://lavalle.pl/planning/node400.html`

Graph-based approach.

Solving (Geometric) Motion Planning Problems

Mapping-based approach.

Cover-based approach.



Graph-based approach.

Solving (Geometric) Motion Planning Problems

Mapping-based approach.

Cover-based approach.

Graph-based approach.

- (1) Find a *graph* of configurations capturing the essence of $\mathcal{Q}_{\text{free}}$
- (2) Find an optimal solution using graph search algorithms

Difference

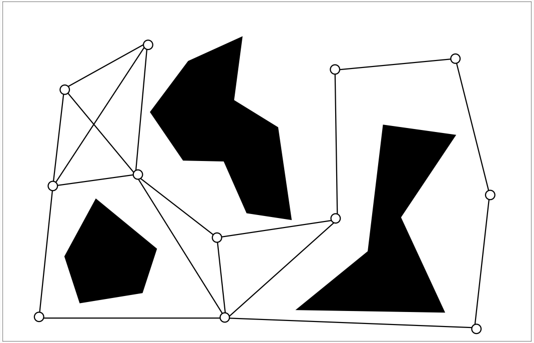
- Difficult to map configuration space if dimension ≥ 4
- Graph-based outperforms mapping-based and cover-based almost everywhere
- We concentrate here exclusively on the graph-based approach

Graph-based Motion Planning

Graph-based Motion Planning

Graph-based Approach

- (1) Find a *graph* of configurations capturing the essence of $\mathcal{Q}_{\text{free}}$
- (2) Find an optimal solution using graph search algorithms



Graph-based Motion Planning

Variants of Graphs

- Graph representation in memory (explicit vs implicit)
- Graph construction (skeletons vs cell decomposition)

Graph-based Motion Planning

Graph Representations: Explicit vs Implicit

Explicit vs Implicit Graphs

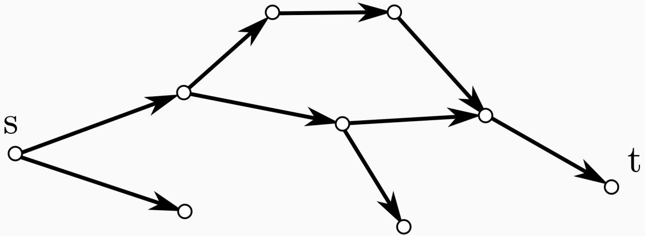
Graph Representation

How we store a graph.

- Explicit graphs: Construct graph (explicitly) in memory, then search over it
- Implicit graphs: Initialize start state, then define successor function

Explicit graph search

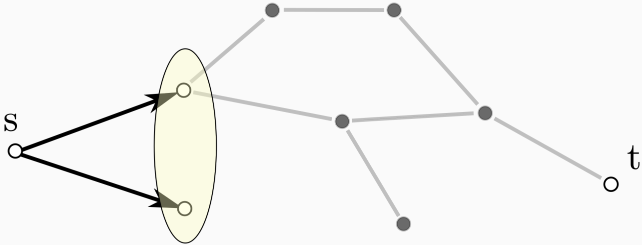
- (1) Create a graph $G = (V, E)$ in memory
- (2) Search graph G



Implicit Graph

Implicit graph search

- (1) Initialize start state s
- (2) Define successor function Γ
- (2) Search graph by expanding the next best node



Graph Representations

Advantages Explicit Graphs

Disadvantages Explicit Graphs

Graph Representations

Advantages Explicit Graphs

- Graphs construction can be computationally expensive
- Fast search (successor function is just a look-up)

Disadvantages Explicit Graphs

- Does not work in infinite spaces
- High memory usage

Graph-based Motion Planning

Graph Construction: Skeletons vs Cell Decomposition

Graph Construction

There are multiple ways to construct graphs

- Skeletonization (From topological skeleton)
  - Visibility Graph
  - Voronoi Diagram
  - Roadmap
  - Random Tree
- Cell decomposition
  - X-connected grids
  - Lattice-based graphs

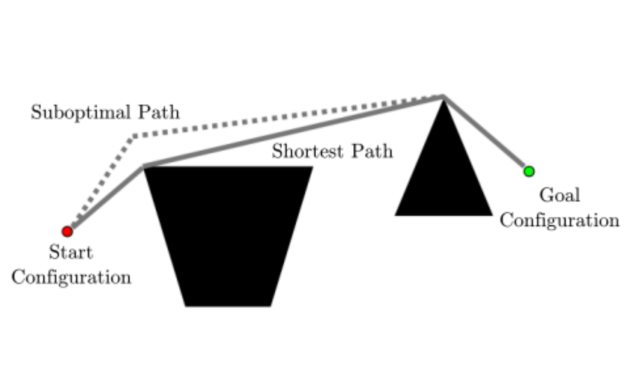
Graph-based Motion Planning

Visibility Graph

Skeletonization: Visibility Graph

Visibility Graphs

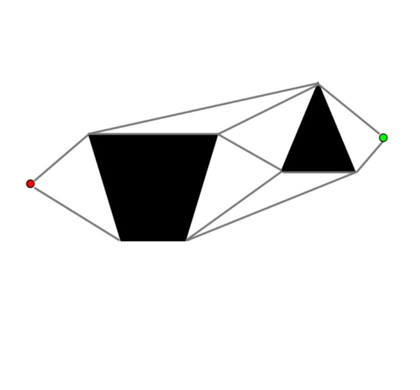
- Idea: Shortest path consists of obstacle-free straight line segments connecting obstacle vertices and initial/goal configuration



Skeletonization: Visibility Graph

Visibility Graphs

- Construct graph by connecting all obstacle vertices + start + goal by straight-line segments (complexity: $O(m^2)$ with m number of vertices).



Skeletonization: Visibility Graph

Advantages

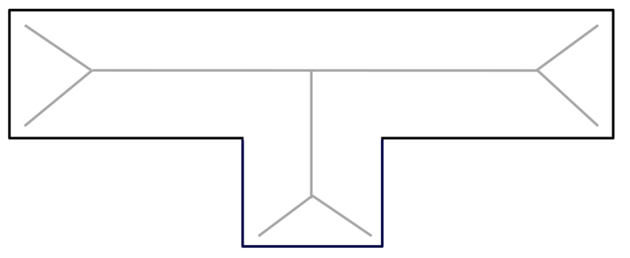
- Independent of dimensionality of configuration space

Disadvantages

- Path might be too close to obstacle
- Cannot deal with non-distance cost
- Cannot deal with non-polygon obstacles
- Requires explicit configuration space obstacles

Voronoi Diagram

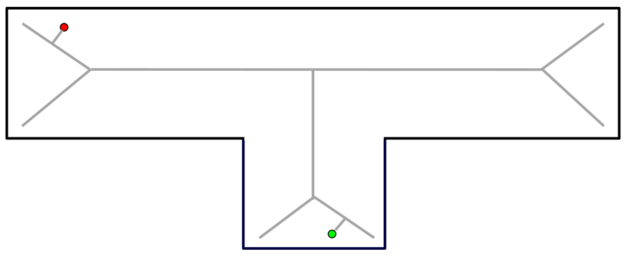
- Idea: Set of all points equidistant to two nearest obstacles (complexity: $O(mlog(m))$ with m number of vertices).



Skeletonization: Voronoi Diagram

Voronoi Diagram

- Construct a graph: Edges as boundaries, vertices as intersection of boundaries
- Add start/goal vertex, and connect them to graph



Skeletonization: Voronoi Diagram

Advantages

- Independent of dimensionality of configuration space
- Stays away from obstacles
- Works with obstacles represented as set of points

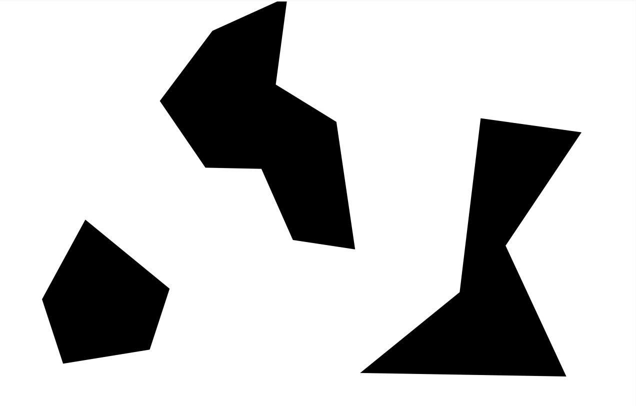
Disadvantages

- Can result in highly suboptimal paths
- Cannot deal with non-distance cost
- Hard to build/maintain in higher dimensions

Skeletonization: Probabilistic Roadmap

Probabilistic Roadmap

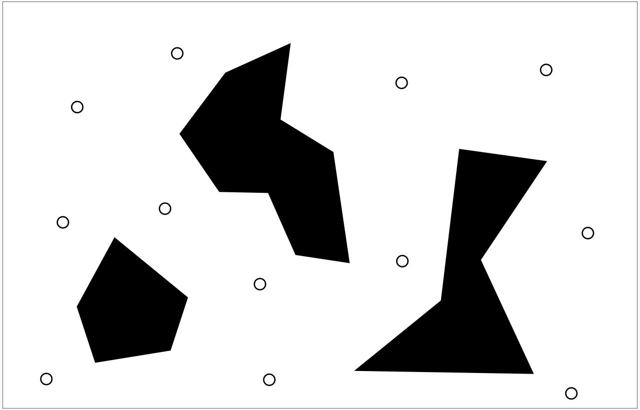
Idea: Sample random points in configuration space



Skeletonization: Probabilistic Roadmap

Probabilistic Roadmap

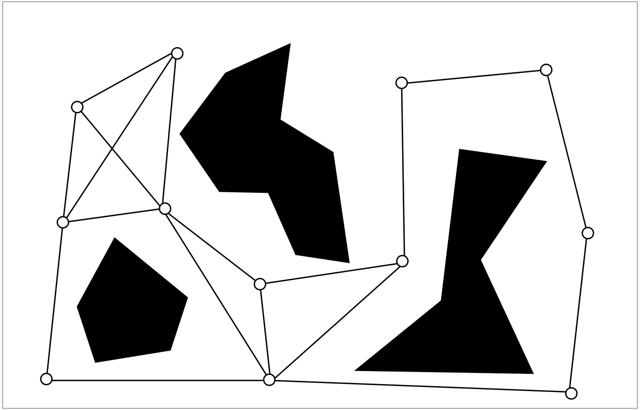
Idea: Sample random points in configuration space



Skeletonization: Probabilistic Roadmap

Probabilistic Roadmap

Idea: Sample random points in configuration space



Skeletonization: Probabilistic Roadmap

Advantages

- Does not require configuration space obstacles (implicit)
- Can quickly discover connected components
- Works with arbitrarily shaped objects

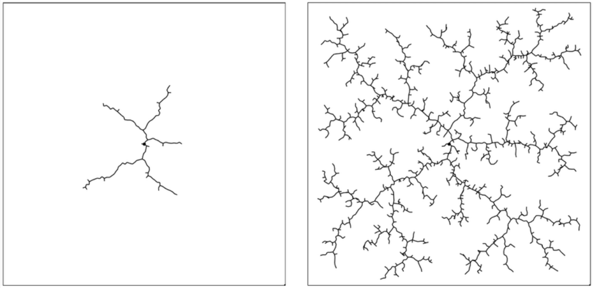
Disadvantages

- Can sample irrelevant portions of configuration space
- Might take long time to find samples in constrained areas

Skeletonization: Random Tree

Random Trees

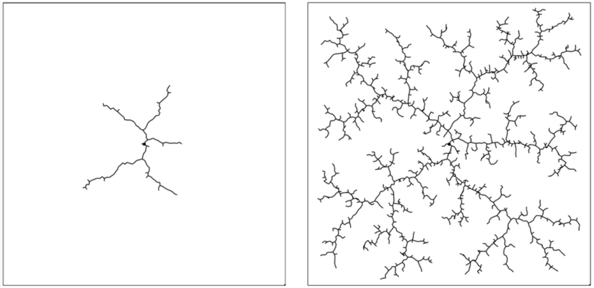
Grow tree from start configuration, walk into random directions



Skeletonization: Random Tree

Random Trees

Grow tree from start configuration, walk into random directions

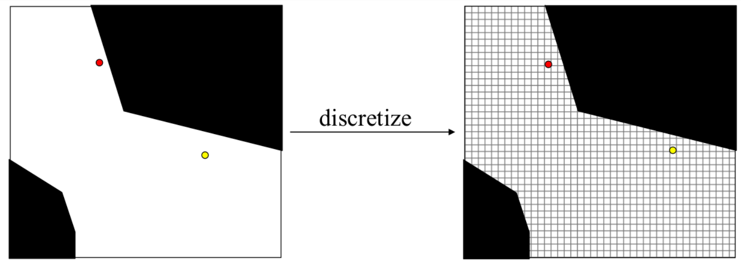


Future lecture!

Cell Decomposition: X-connected Grids

X-connected grids

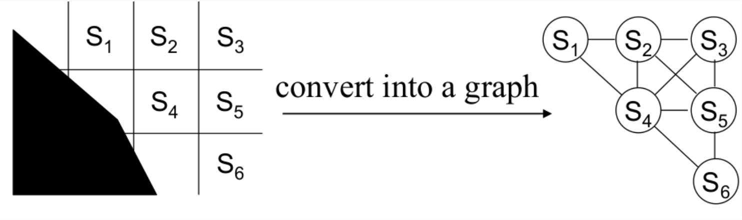
Add uniform grid onto configuration space (discretization)



X-connected Grids

Every grid cell in free space is a node, two nodes are connected if they are neighbors

- 4-connected: only horizontal/vertical connections (up to 4 neighbors per cell)
- 8-connected: allow diagonal connections (up to 8 neighbors per cell)



Skeletonization: X-connected Grids

Advantages

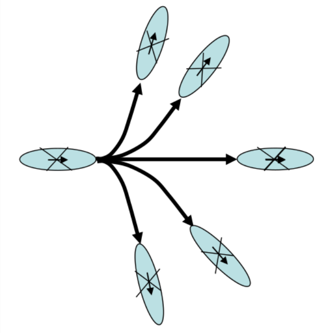
- Simple to implement
- Can deal with arbitrarily shaped obstacles
- Works with any cost function

Disadvantages

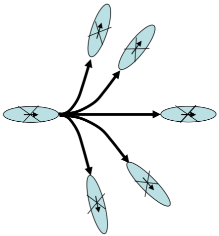
- Scales badly with number of dimensions (10 dimensions, 100 discretizations per dimension $\Rightarrow 100^{10}$ grid cells)

Cell Decomposition: Lattice-based Graphs

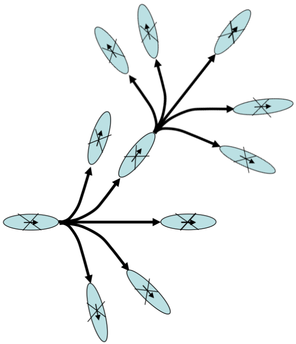
Idea: Define nodes as transitions from previous nodes. Example: Use precomputed motion primitives to expand a node.



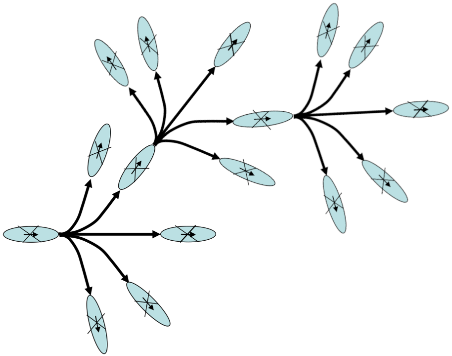
Cell Decomposition: Lattice-based Graphs



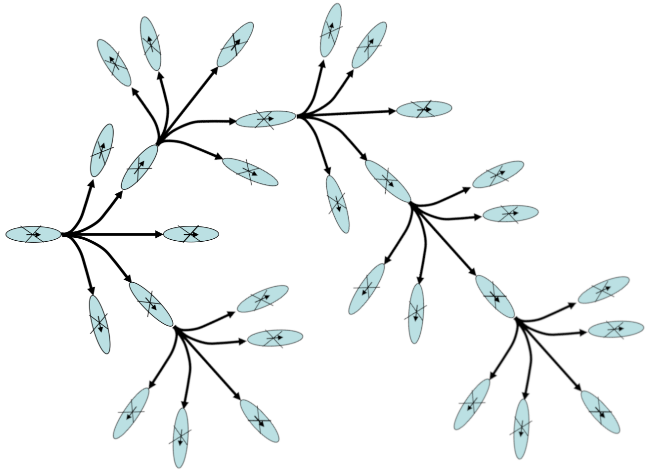
Cell Decomposition: Lattice-based Graphs



Cell Decomposition: Lattice-based Graphs

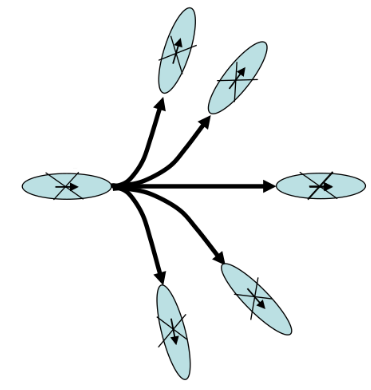


Cell Decomposition: Lattice-based Graphs



Cell Decomposition: Lattice-based Graphs

Idea: Define nodes as transitions from previous nodes. Use precomputed motion primitives to expand a node.



Next lecture!

Graph-based planning: Recap

Summary

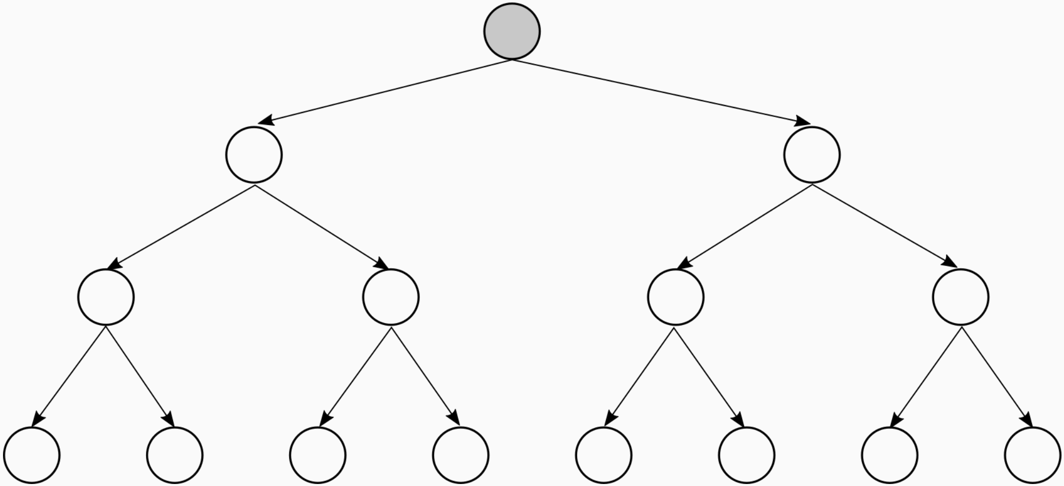
- Mapping-, Cover-, Graph-based
- Graph representation (Explicit vs Implicit)
- Graph construction (Skeletonization vs Cells)

Graph Search

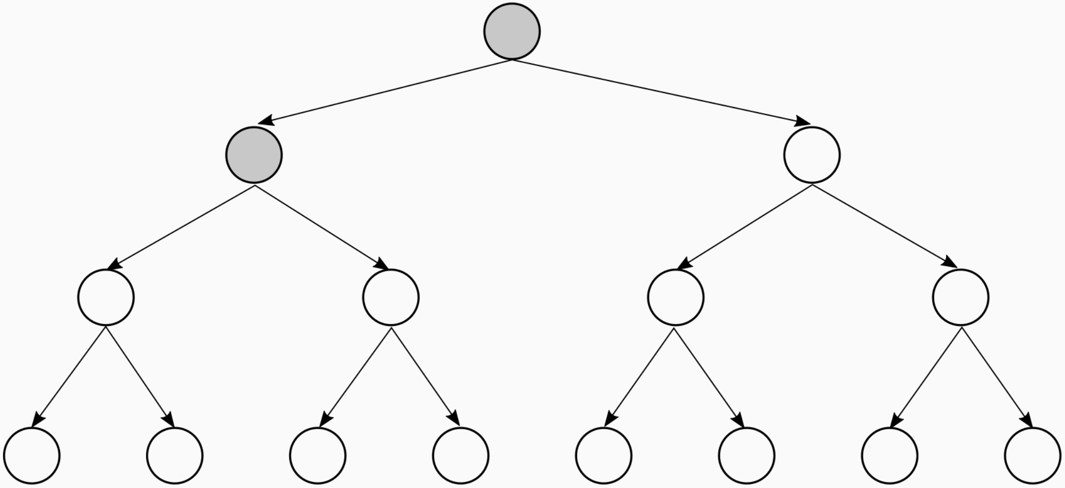
Graph Search

- Uninformed search
  - Depth-first search (DFS)
  - Breadth-first search (BFS)
  - Dijkstra
- Uses current path cost, but no estimate of distance to goal

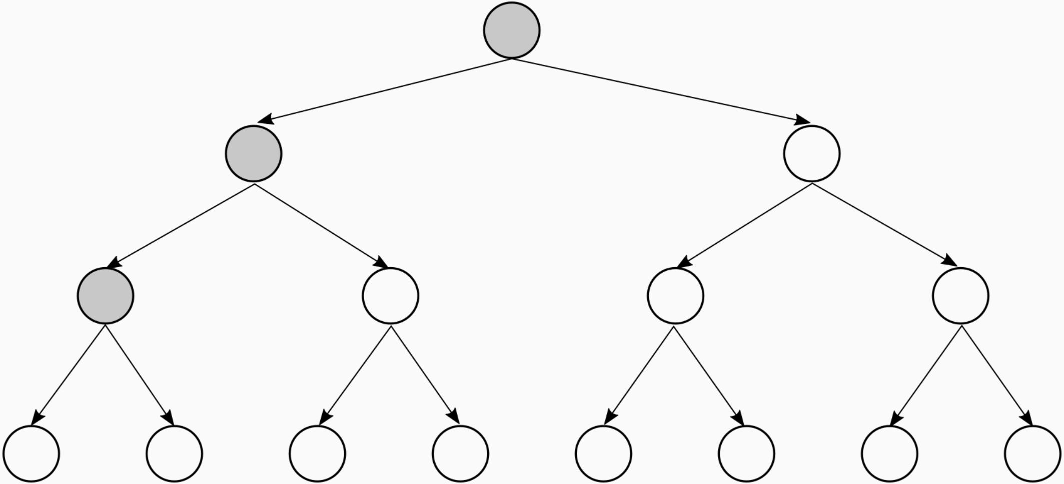
Graph Search: Depth-first Search



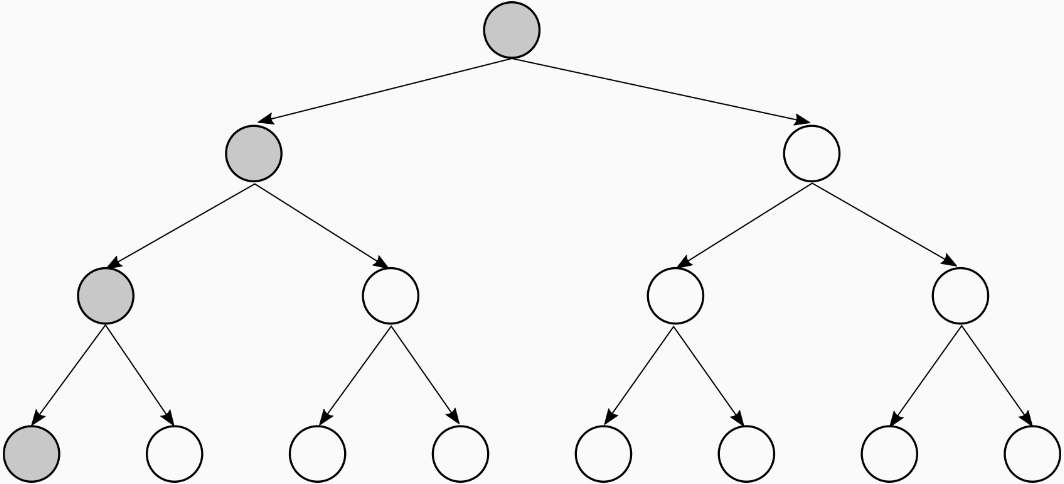
Graph Search: Depth-first Search



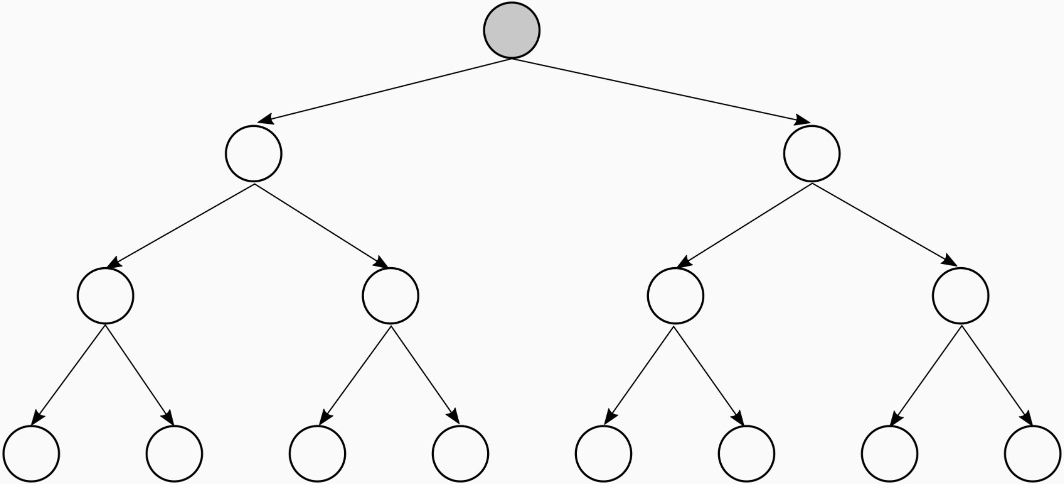
Graph Search: Depth-first Search



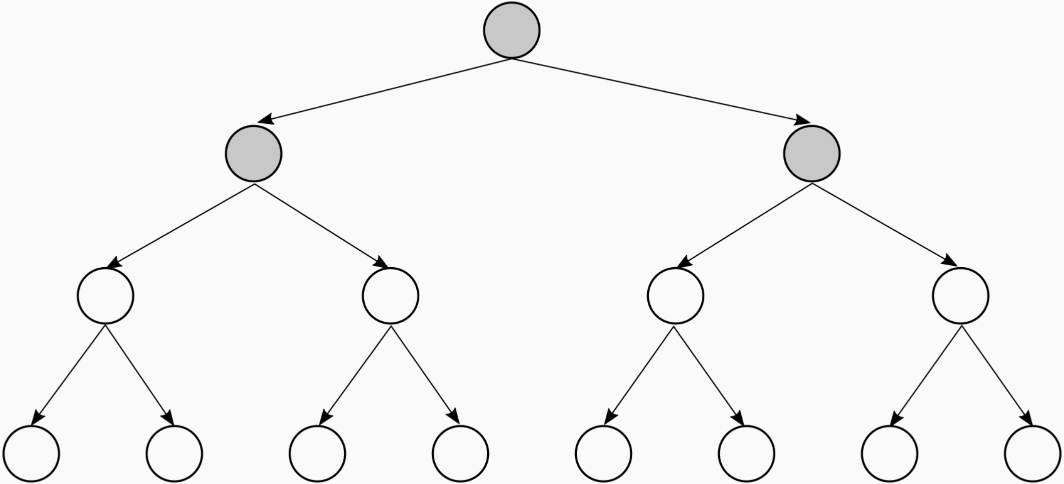
Graph Search: Depth-first Search



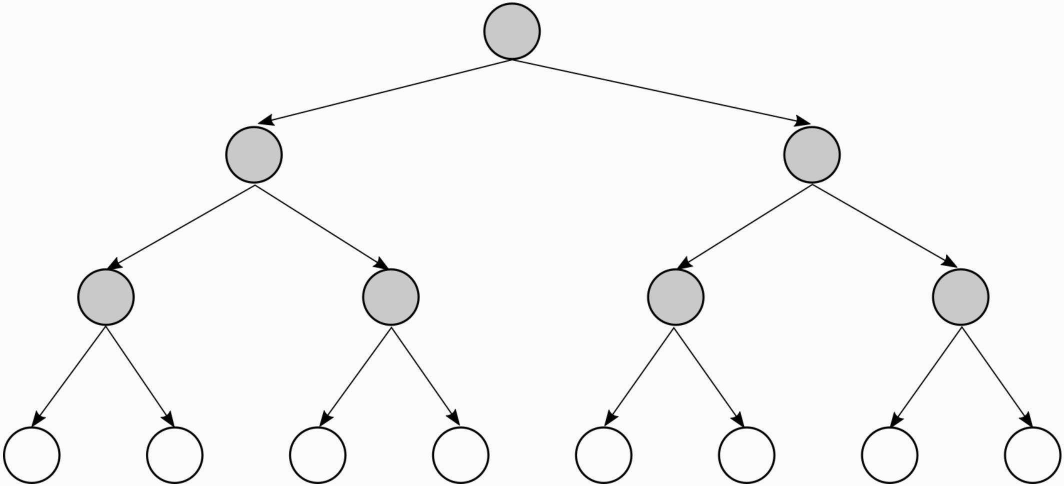
Graph Search: Breadth-first Search



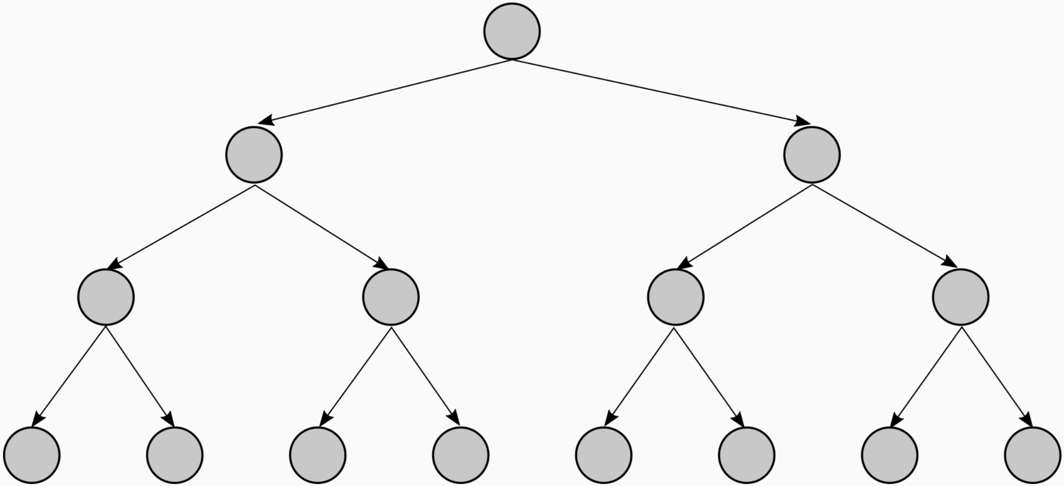
Graph Search: Breadth-first Search



Graph Search: Breadth-first Search



Graph Search: Breadth-first Search



Graph Search

Graph Search

Graph Search

Informed Graph Search

Idea: Inject domain knowledge into search algorithm.

Trade-off between computation time of this knowledge and runtime of search.

A* Search

A* Algorithm

- Informed graph search algorithm
- Paper: "A formal basis for the heuristic determination of minimum cost paths", by PE Hart, NJ Nilsson, B Raphael (1968), Cited by 15300 (google scholar, 05/2024).

A*: Main properties

- Complete: Finds the solution of one exists (and report if none exists)
- Optimal: Finds the solution with the lowest cost
- Optimal efficient: No other algorithm can search better, given the heuristic function

Lies at the foundation of almost every motion planning algorithm out there

A* Topics

- Notations and datastructures
- Pseudocode and example
- Proof of optimality

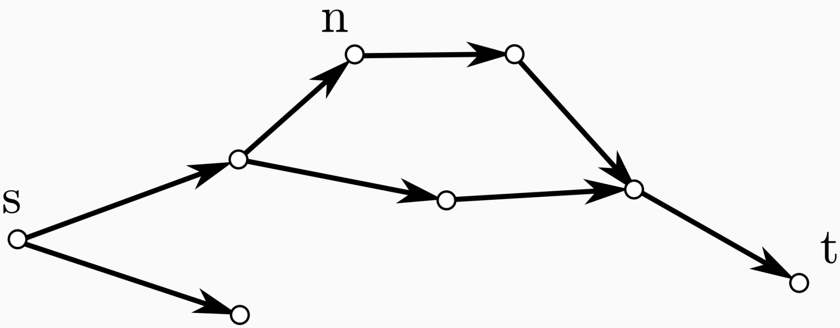
A* Search

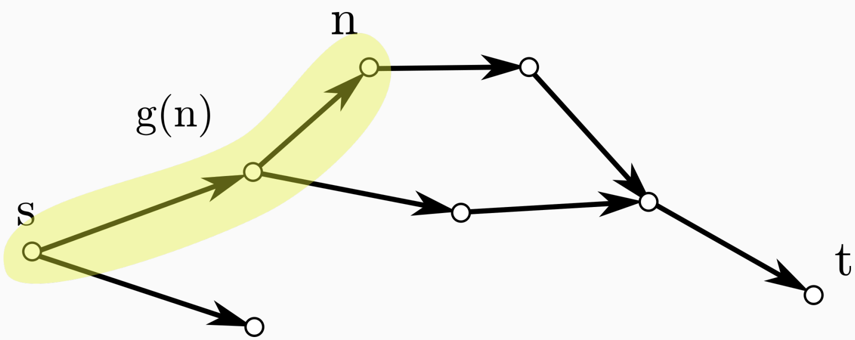
Notations and Datastructures

Notations and Datastructures

- Graph $G = (V, E)$ with $V = \{n_i\}$, $E = \{e_{ij}\}$ with costs c_{ij}
- Start node $s \in V$
- Goal node $t \in V$
- Successor function Γ, taking n_i as input and generating $\{(n_j, c_{ij})\}$

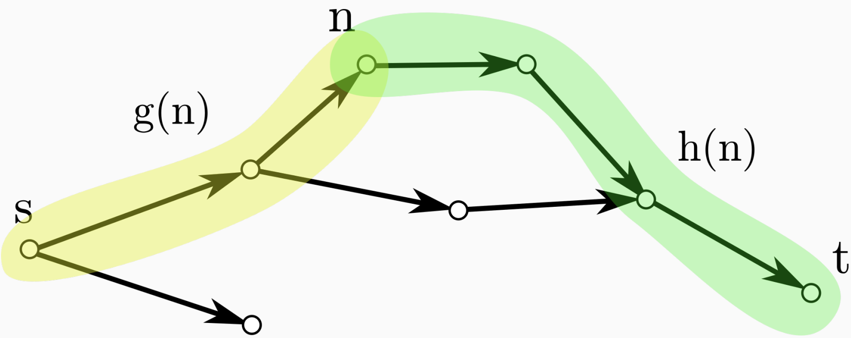
- Cost-to-go $h(n)$: Cost of optimal path from node n to t
- Cost-to-come $g(n)$: Cost of optimal path from s to node n
- Total cost $f(n) = g(n) + h(n)$: Cost of optimal path from s to t going through node n


n
s
t



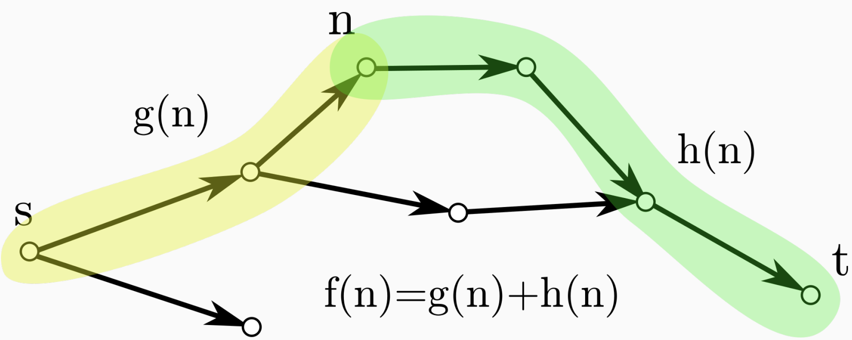
Cost-to-come g

$g(n)$: Cost of optimal path from s to n.



Cost-to-go h

$h(n)$: Cost of optimal path from n to t.



Total cost f

$f(n)$: Cost of optimal path from s to t, constrained by n.

A* Idea

- Cost terms h, f are unknown
- Use estimate function $\hat{f}(n) = g(n) + \hat{h}(n)$
- $\hat{h}(n)$ is called the heuristic function — needs to be consistent and/or admissible for optimality to hold

Consistent/Monotone Heuristic

$$h(x) \leq h(y) + d(x, y)$$

Admissible Heuristic

Never overestimate the cost to reach the goal, i.e.:

$$\hat{h}(n) \leq h(n)$$

Relationship admissible and consistent heuristics as part of exercise!

A* Search

A* Pseudocode and Example

Pseudocode A*

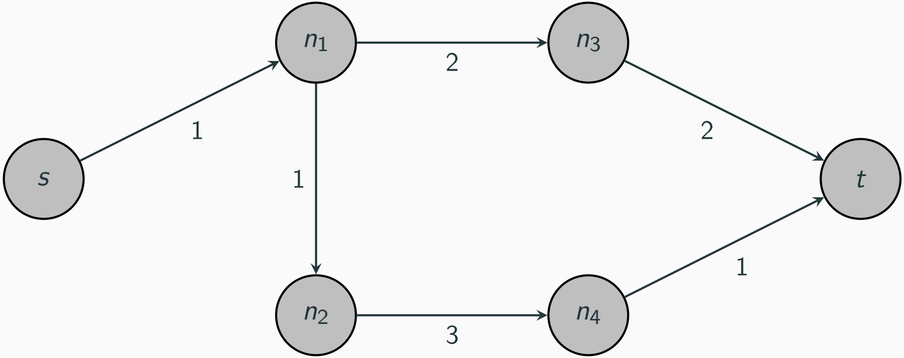
- $\text{OPEN} = \{s\}$, $\text{CLOSED} = \emptyset$
- While true:
    1. If OPEN is $\emptyset$: Exit with **failure**
    2. $n \leftarrow$ Remove minimal $\hat{f}(n)$ node from OPEN
    3. If n is goal node: Add n to CLOSED, construct path and exit with **success**.
    4. $\textsc{ExpandNode}(n)$

Pseudocode A*

- OPEN $= \{s\}$, CLOSED $= \emptyset$
- While true:
  1. If OPEN is $\emptyset$: Exit with **failure**
  2. $n \leftarrow$ Remove minimal $\hat{f}(n)$ node from OPEN
  3. If n is goal node: Add n to CLOSED, construct path and exit with **success**.
  4. Add n to CLOSED

     For each n' in $\textsc{Successors}(n)$:
     - If n' is in closed: continue
     - $s = g(n) + c(n, n')$
     - If $s < g(n')$:
     - $\quad g(n') = s$
     - $\quad f(n') = g(n') + h(n')$
     - $\quad$ If n' is in OPEN:
     - $\qquad$ Update n' in OPEN
     - $\quad$ Else:
     - $\qquad$ Add n' to OPEN

A* Example

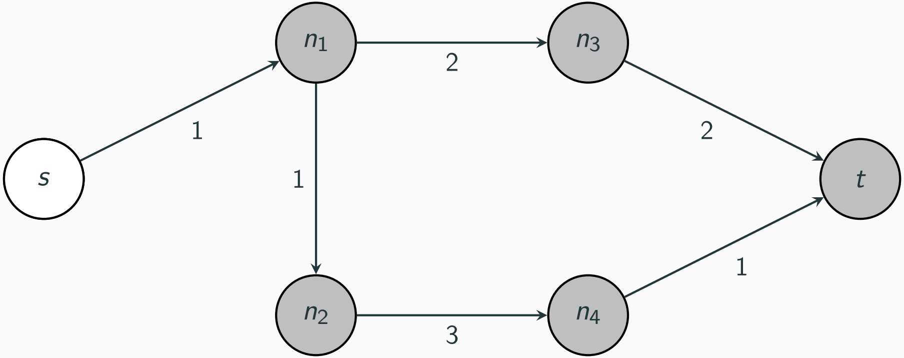


$g(s) = 0, g(n_1) = \infty, g(n_2) = \infty, g(n_3) = \infty, g(n_4) = \infty, g(t) = \infty$

$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$

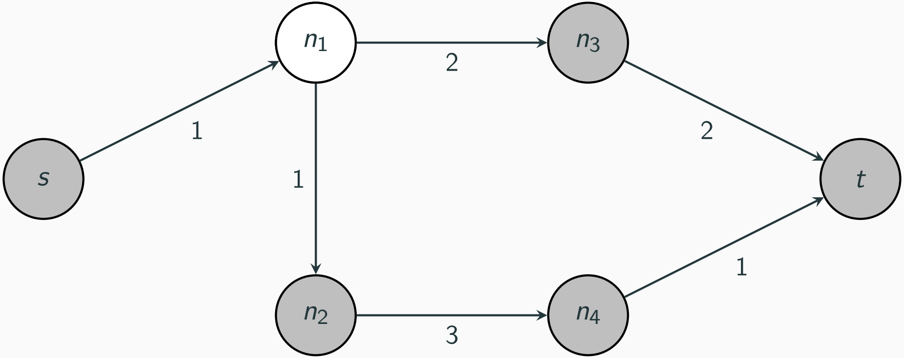
OPEN $= \{\}$ Next state to expand:

A* Example



$g(s) = 0, g(n_1) = \infty, g(n_2) = \infty, g(n_3) = \infty, g(n_4) = \infty, g(t) = \infty$
$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$
OPEN $= \{\mathbf{s}\}$ Next state to expand: s

A* Example

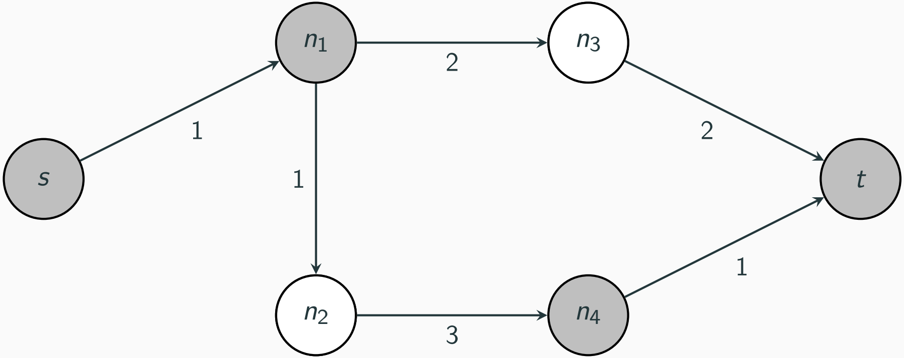


$g(s) = 0, \mathbf{g(n_1) = 1}, g(n_2) = \infty, g(n_3) = \infty, g(n_4) = \infty, g(t) = \infty$

$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$

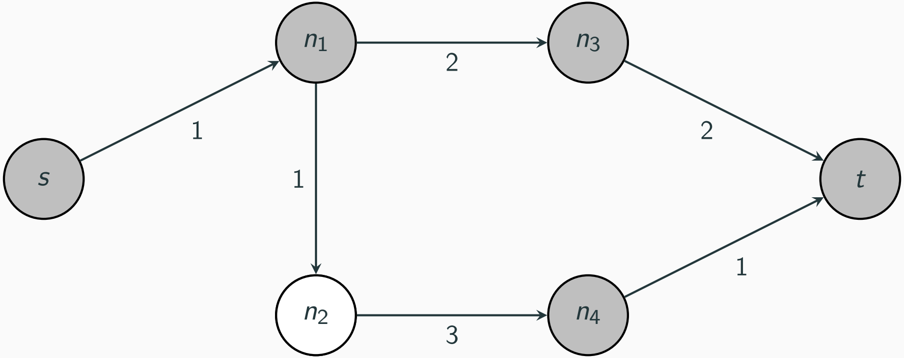
OPEN $= \{n_1\}$ Next state to expand: n_1

A* Example



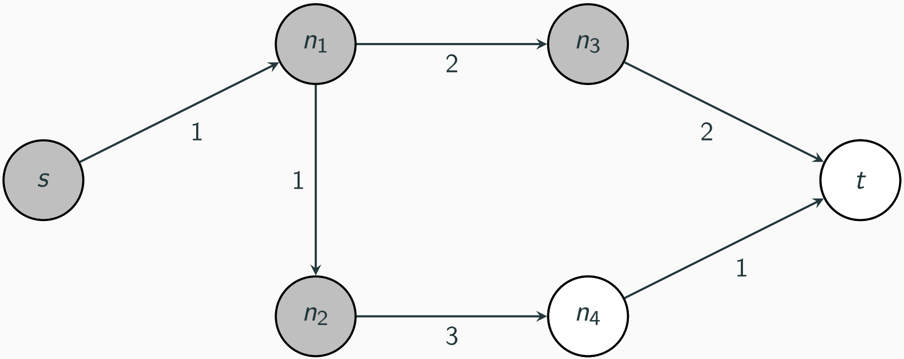
$g(s) = 0, g(n_1) = 1, \mathbf{g(n_2) = 2}, \mathbf{g(n_3) = 3}, g(n_4) = \infty, g(t) = \infty$
$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$
OPEN $= \{n_2, n_3\}$ Next state to expand: n_3

A* Example



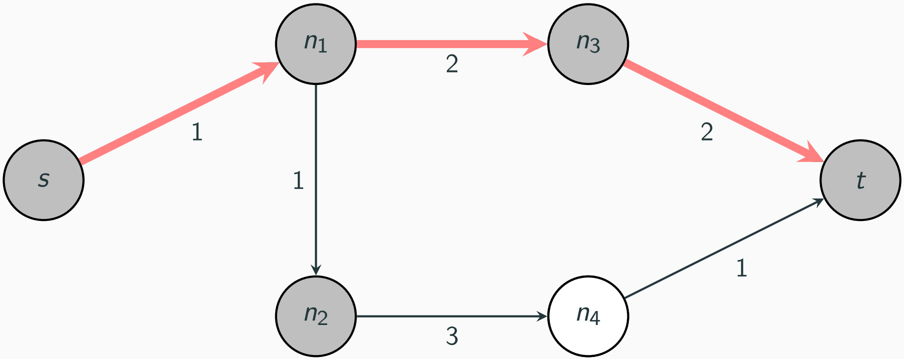
$g(s) = 0, g(n_1) = 1, g(n_2) = 2, g(n_3) = 3, g(n_4) = \infty, g(t) = 5$
$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$
OPEN $= \{n_2, t\}$ Next state to expand: n_2

A* Example



$g(s) = 0, g(n_1) = 1, g(n_2) = 2, g(n_3) = 3, \mathbf{g(n_4)} = \mathbf{5}, g(t) = 5$
$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$
OPEN $= \{t, n_4\}$ Next state to expand: t

A* Example



$g(s) = 0, g(n_1) = 1, g(n_2) = 2, g(n_3) = 3, g(n_4) = 5, g(t) = 5$
$\hat{h}(s) = 3, \hat{h}(n_1) = 2, \hat{h}(n_2) = 2, \hat{h}(n_3) = 1, \hat{h}(n_4) = 1, \hat{h}(t) = 0$
OPEN $= \{n_4\}$ Next state to expand:

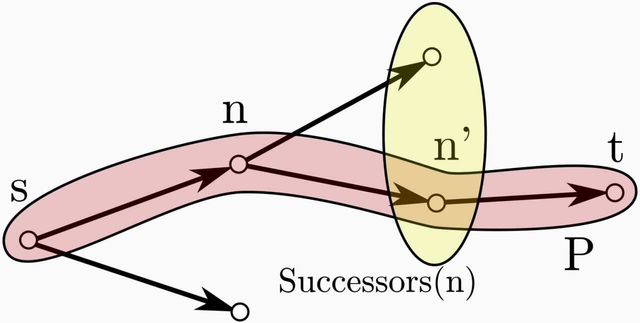
Optimality of A*

Properties of A*

- A* is complete and optimal (it will return the optimal path, if one exists) (sometimes also termed admissible)
- A* is optimal efficient (it performs the minimal number of state expansions)

Let us prove, that A* is complete and optimal (it is guaranteed to find an optimal path from s to t if one exists).

- Lemma 1: A* maintains always a node in OPEN, which lies on optimal path.
- Lemma 2: If h is admissible, then A* always maintains a node which underestimates the evaluation cost f.
- Theorem: If h is admissible, then A* is complete and optimal.



Lemma 1

A* always maintains at least one node from optimal path P in OPEN.

Lemma 1

There exist *always* a node n' on an optimal path P in OPEN with $\hat{g}(n') = g(n')$ (if one exists).

Proof

Let $P = \{s = n_0, n_1, \ldots, n_k = t\}$ be an optimal path.

Case 1: s is in OPEN. Then $\hat{g}(s) = g(s) = 0$ and $n' = s$ is an open node on P.

Case 2: If s is not in OPEN. Let Δ be all nodes in CLOSED lying on P, and let n be the largest index of them. Since n is in CLOSED, it has (1) been expanded, and (2) there exists a successor node n' on P which is in OPEN. Therefore $\hat{g}(n') = g(n')$. $\square$

Lemma 2

If $\hat{h}(n') \leq h(n)$ (admissible heuristic), then there exist *always* a node n' on an optimal path P in OPEN with $\hat{f}(n') \leq f(s)$.

Proof

By Lemma 1, there exists an open node n' on P, such that $\hat{g}(n') = g(n')$. By definition of $\hat{f}$:

$$\begin{aligned}
\hat{f}(n') &= \hat{g}(n') + \hat{h}(n') && \text{(by Definition)} \\
&= g(n') + \hat{h}(n') && \text{(by Lemma 1)} \\
&\leq g(n') + h(n') && \text{(by Admissibility)} \\
&= f(n') = f(s) && \text{(Since } n' \text{ is on } P\text{)}
\end{aligned}$$

Therefore $\hat{f}(n') \leq f(s)$. $\square$

Theorem 1

If $\hat{h}(n') \leq h(n)$ (admissible heuristic), A* is admissible. [a]

[a] An algorithm is admissible, if it is guaranteed to find an optimal solution if one exists.

Proof

Proof by contradiction. Assumption: There exists an optimal path and A* terminates, but A* does NOT find the optimal path.

- A* terminates at non-goal node: Cannot happen, since we terminate at a goal node by definition.
- A* terminates at goal node, but along non-optimal path.
  (1) Assume A* terminates at goal node t but $\hat{f}(t) = \hat{g}(t) > f(s)$.
  (2) But, by Lemma 2, before termination, there must have been n' in open, such that $\hat{f}(n') \leq f(s) < \hat{f}(t)$.
  (3)Then, by definition, A* would have expanded n', not terminated at t. A contradiction. $\square$

Optimal and Complete

A* is admissible: It will find the optimal solution if it exists.

Best Informed Search

A* is also optimal in another sense: It will expand the least amount of nodes (proof, see Hart, Nilsson, and Raphael (1968)).

Advantage of A*

There is (provably) no better algorithm with respect to the knowledge available (i.e. the admissible heuristic).

Admissible Heuristics

- Admissible heuristics is a cost-to-go estimate, which always underestimates real cost-to-go
- Similar to lower bounds in optimization
- Allows us to prune away many nodes in a search problem (while guaranteeing optimality!)

But: How do we actually find admissible heuristics?

"Admissible heuristics as solutions to simplified problems"

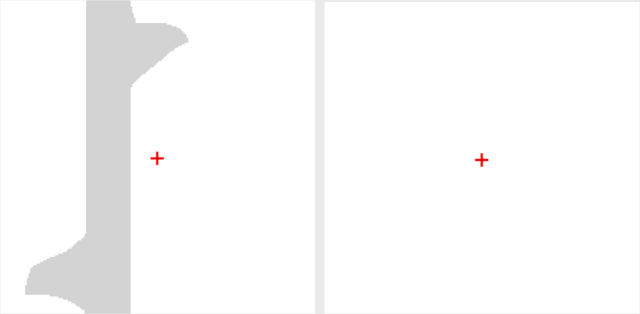
—*Judea Pearl - Heuristics (1984)*

- Solution to relaxed problem (less constraints)

"Admissible heuristics as solutions to simplified problems"

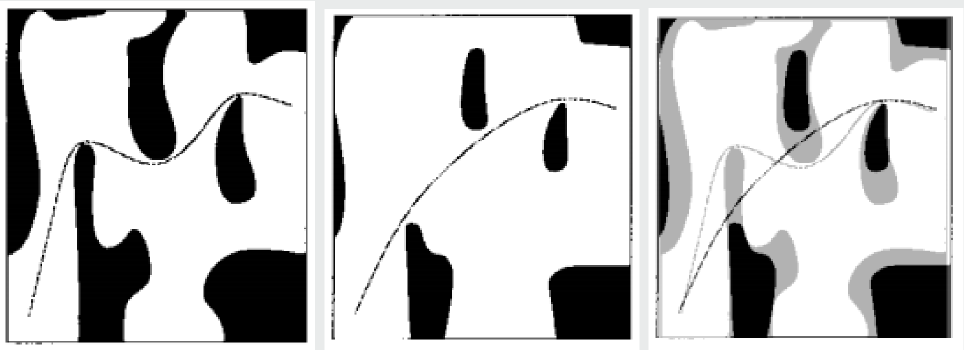
—*Judea Pearl - Heuristics (1984)*

- Example: Remove obstacles (free space assumption) (metric!)



"Admissible heuristics as solutions to simplified problems"
—*Judea Pearl - Heuristics (1984)*

- Example: Shrink obstacles

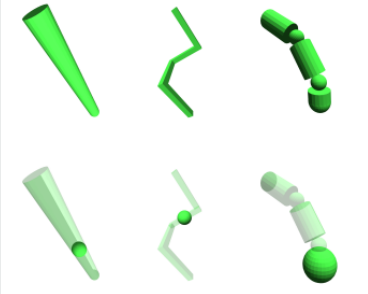


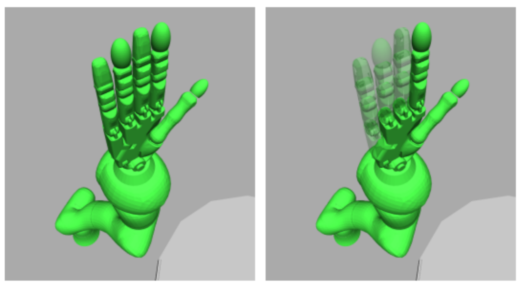
"A method of progressive constraints for manipulation planning", Ferbach and Barraquand (1997)

"Admissible heuristics as solutions to simplified problems"

—*Judea Pearl - Heuristics (1984)*

- Example: Remove joints (multilevel motion planning)



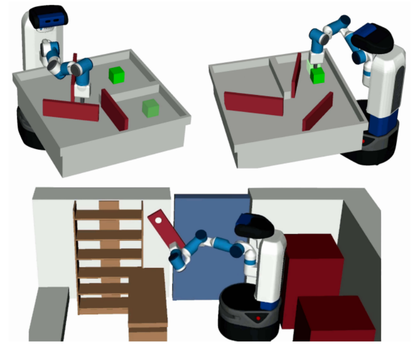


"Section Patterns: Efficiently Solving Narrow Passage Problems in Multilevel Motion Planning", Orthey and Toussaint (2021)

"Admissible heuristics as solutions to simplified problems"

—*Judea Pearl - Heuristics (1984)*

- Example: Precompute obstacle motions (Factored state spaces)



"Solving Rearrangement Puzzles using Path Defragmentation in Factored State Spaces", Bayraktar et al. (2023)

Conclusion

Summary

- Finding good representations of configuration space
  - Explicit vs Implicit graphs
  - Skeletons vs Cell decomposition
- A* to search over graphs
- Optimality proof of A* search
- Admissible heuristics

Suggested Reading

- Representations, Informed search, and A*
  https://www.cs.cmu.edu/~maxim/classes/robotplanning_grad/ (Maxim Likhachev)
- A* paper (Hart, Nilsson, and Raphael, 1968)
- Admissible heuristics (Pearl, 1984)

References i

[1] Tomás Lozano-Pérez and Michael A. Wesley. "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles". In: *Commun. ACM* 22.10 (Oct. 1979), pp. 560–570. ISSN: 0001-0782. DOI: 10.1145/359156.359164.

[2] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[3] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. 1984.

[4] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006. ISBN: 978-0-521-86205-9. URL: http://planning.cs.uiuc.edu.