R.O.B.O.T. Comics

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

Motion Planning

METR 4202: **Robotics** & Automation

Dr Surya Singh -- Lecture # 10

October 4, 2017

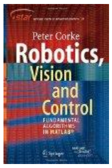
metr4202-staff@itee.uq.edu.au

http://robotics.itee.uq.edu.au/~metr4202/ [http://**metr4202.com**]

Lecture Schedule

Week	Date	Lecture (W: 3:05p-4:50, 7-222)
1	26-Jul	Introduction Representing Position & Orientation & State
2	2-Aug	Robot Forward Kinematics (Frames, Transformation Matrices & Affine Transformations)
3	9-Aug	Robot Inverse Kinematics & Dynamics (Jacobians)
4	16-Aug	*Ekka Day* (Robot Kinematics & Kinetics Review)
5	23-Aug	Jacobians & Robot Sensing Overview
6	30-Aug	Robot Sensing: Single View Geometry & Lines
7	6-Sep	Robot Sensing: Basic Feature Detection
8	13-Sep	Robot Sensing: Scalable Feature Detection
9	20-Sep	Mid-Semester Exam & Multiple View Geometry
	27-Sep	*Study break*
10	4-Oct	**Motion Planning**
11	11-Oct	Probabilistic Robotics: Planning & Control (Sample-Based Planning/State-Space/LQR)
12	18-Oct	Probabilistic Robotics: Localization & SLAM
13	25-Oct	The Future of Robotics/Automation + Challenges + Course Review

Follow Along Reading:



[Robotics, Vision & Control](#)
by [Peter Corke](#)

Also online:[SpringerLink](#)

[UQ Library eBook: 364220144X](#)

Today

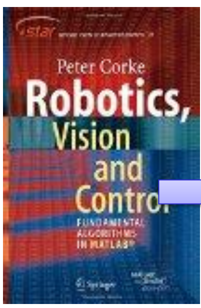
➔Planning & Control ⇚

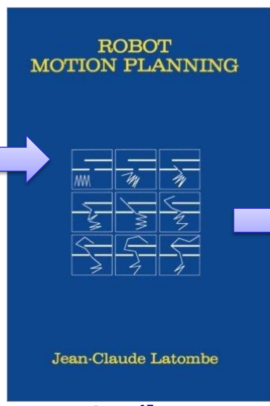
- Planning (**<u>Global</u>** Motion)
  - pp. 91-103
    (Yup! That's all Peter Corke has to say! Yet there is a Chapter [15] on Visual Servoing, a **<u>local</u>** motion method that can't handle obstacles).

- In Two Weeks: SLAM
  - pp. 123-4
    (§6.4-6.5)

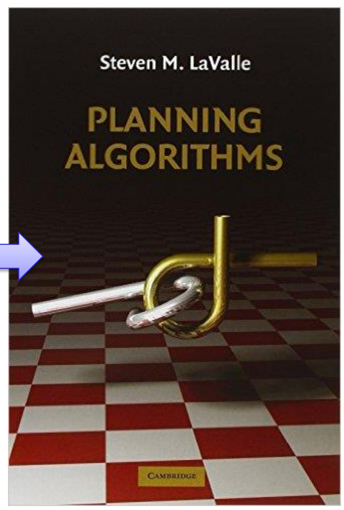
Next Time

Reference Material





[**UQ Library**](#)
(TJ211.4 .L38 1991)



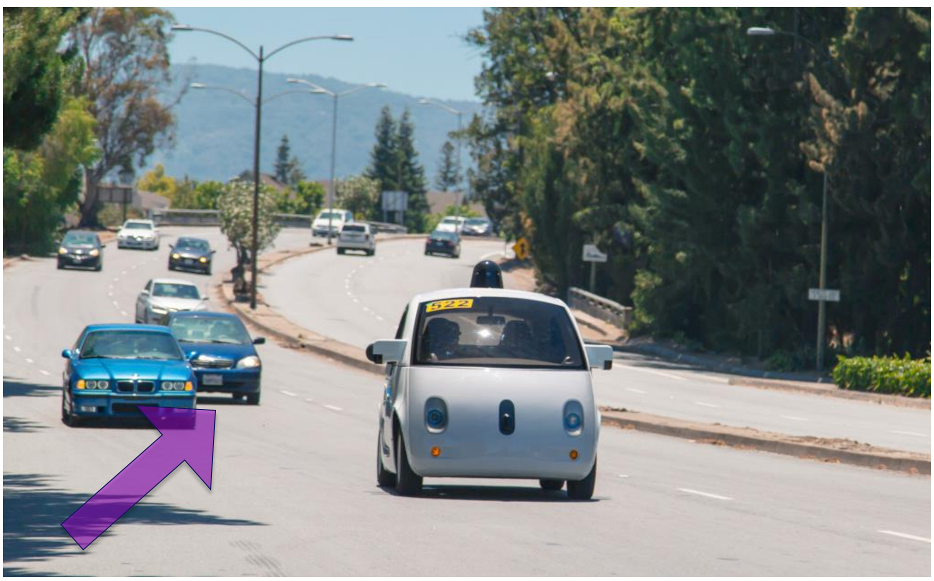
[**UQ Library / Online (PDF)**](#)

(Kinematic) Motion Planning

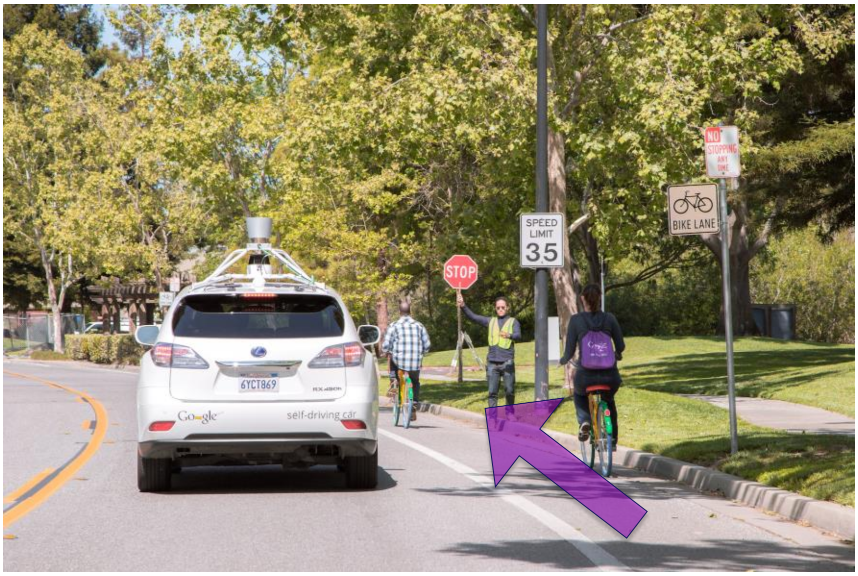
Motion Planning? Let's Get Moving…

Motion Planning? Let's Get Moving?

Motion Planning? The clutter can not be "ignored"



Motion Planning: Processing the Limits

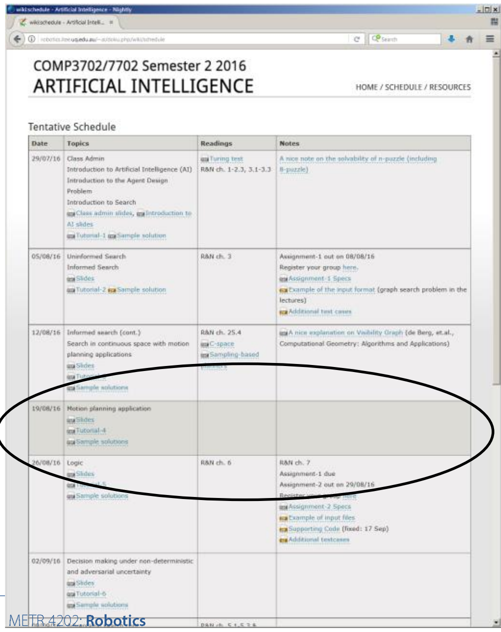


Path-Planning Approaches

- Roadmap
  Represent the connectivity of the free space by a network of 1-D curves
- Cell decomposition
  Decompose the free space into simple cells and represent the connectivity of the free space by the adjacency graph of these cells
- Potential field
  Define a function over the free space that has a global minimum at the goal configuration and follow its steepest descent

Slide from Latombe, CS326A

See Also: http://robotics.itee.uq.edu.au/~ai/



COMP3702/7702 Semester 2 2016

ARTIFICIAL INTELLIGENCE

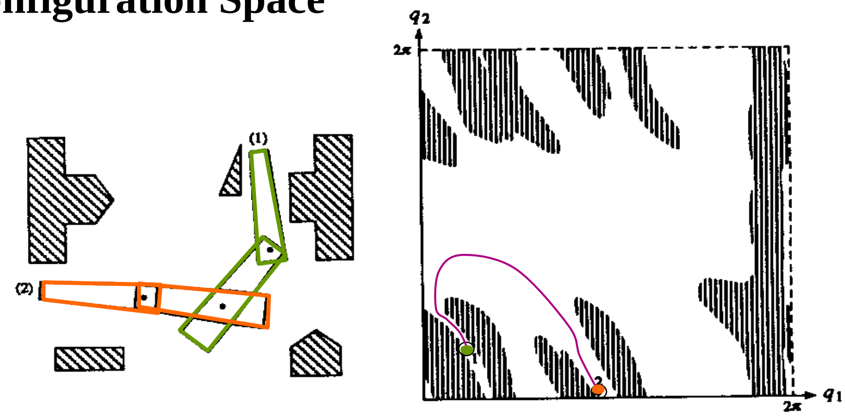
HOME / SCHEDULE / RESOURCES

Tentative Schedule

Date	Topics	Readings	Notes
29/07/16	Class Admin Introduction to Artificial Intelligence (AI) Introduction to the Agent Design Problem Introduction to Search Class admin slides, Introduction to AI slides Tutorial-1 Sample solution	Turing test R&N ch. 1-2.3, 3.1-3.3	A nice note on the solvability of n-puzzle (including 8-puzzle)
05/08/16	Uninformed Search Informed Search Slides Tutorial-2 Sample solution	R&N ch. 3	Assignment-1 out on 08/08/16 Register your group here. Assignment-1 Specs Example of the input format (graph search problem in the lectures) Additional test cases
12/08/16	Informed search (cont.) Search in continuous space with motion planning applications Slides Tutorial-3 Sample solutions	R&N ch. 25.4 C-space Sampling-based planners	A nice explanation on Visibility Graph (de Berg, et.al., Computational Geometry: Algorithms and Applications)
19/08/16	Motion planning application Slides Tutorial-4 Sample solutions		
26/08/16	Logic Slides Tutorial-5 Sample solutions	R&N ch. 6	R&N ch. 7 Assignment-1 due Assignment-2 out on 29/08/16 Register your group here Assignment-2 Specs Example of input files Supporting Code (fixed: 17 Sep) Additional textcases
02/09/16	Decision making under non-deterministic and adversarial uncertainty Slides Tutorial-6 Sample solutions		

External Configuration Is Important …

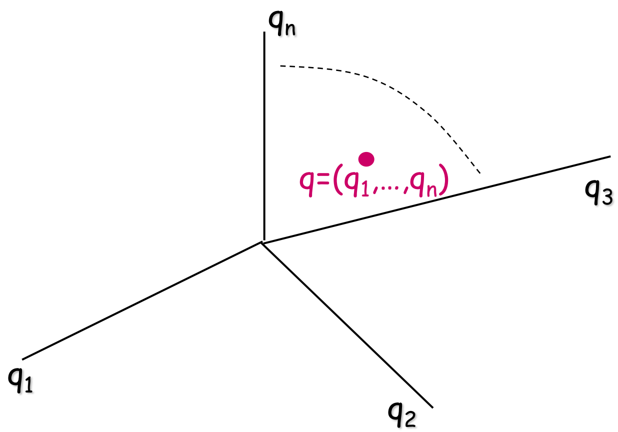
Configuration Space



- A robot configuration is a specification of the positions of all robot points relative to a fixed coordinate system
- Usually a configuration is expressed as a "vector" of position/orientation parameters

Slide from Latombe, CS326A

Motion Planning in C-Space

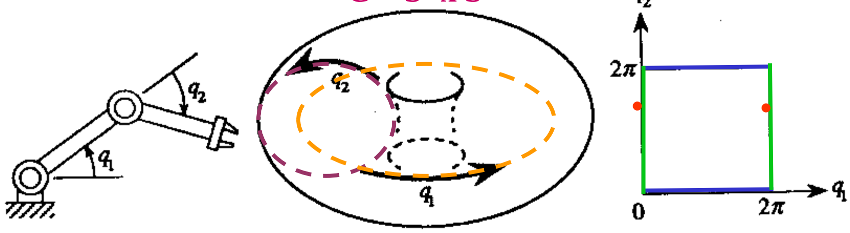


Slide from Latombe, CS326A

Configuration Space of a Robot

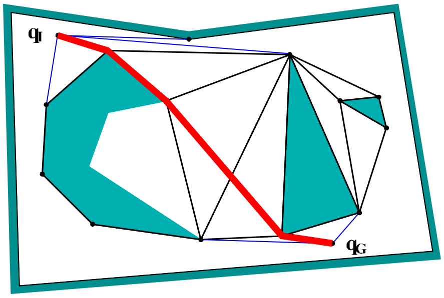
- Space of all its possible configurations
- But the topology of this space is usually not that of a Cartesian space

$C = S^1 \times S^1$



Slide from Latombe, CS326A

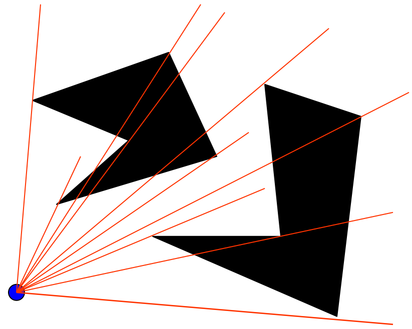
Geometric Planning Methods



- Several Geometric Methods:
  - Vertical (Trapezoidal) Cell Decomposition
  - **Roadmap Methods**
    - **Cell (Triangular) Decomposition**
    - Visibility Graphs
    - Veroni Graphs

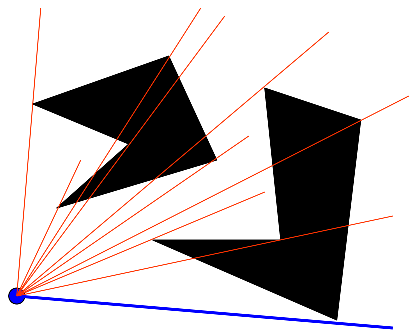
Artwork from LaValle, Ch. 6

I. Rotational Sweep



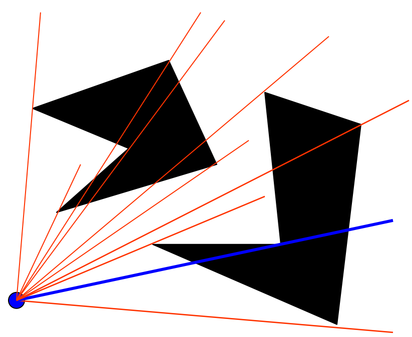
Slide from Latombe, CS326A

Rotational Sweep



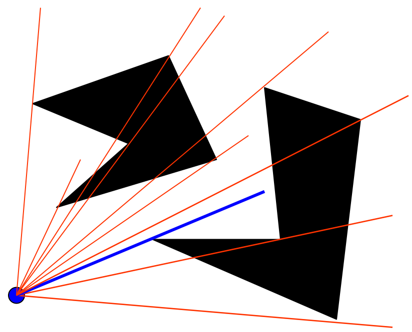
Slide from Latombe, CS326A

Rotational Sweep



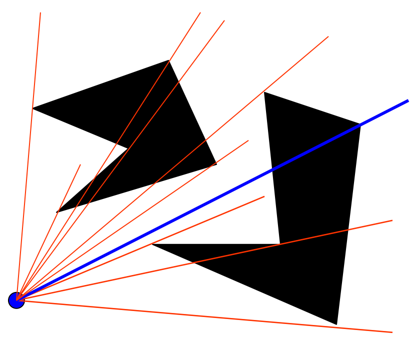
Slide from Latombe, CS326A

Rotational Sweep



Slide from Latombe, CS326A

Rotational Sweep



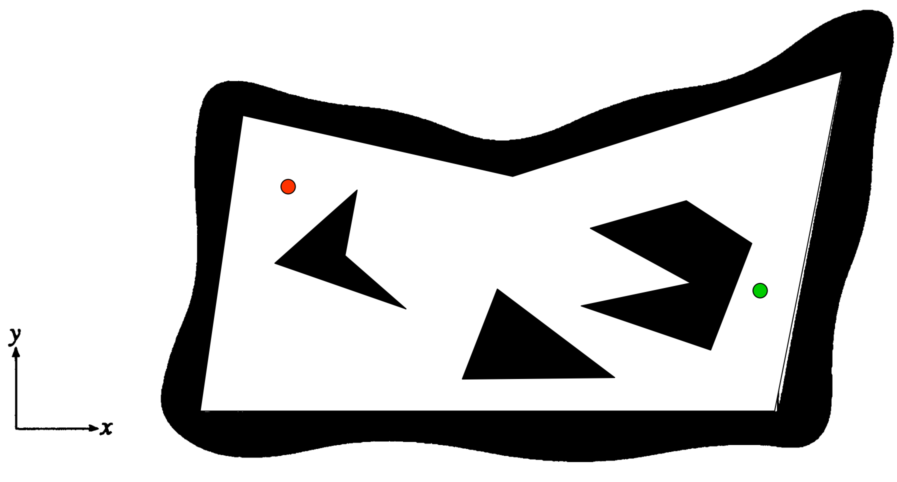
Slide from Latombe, CS326A

II. Cell-Decomposition Methods

Two classes of methods:

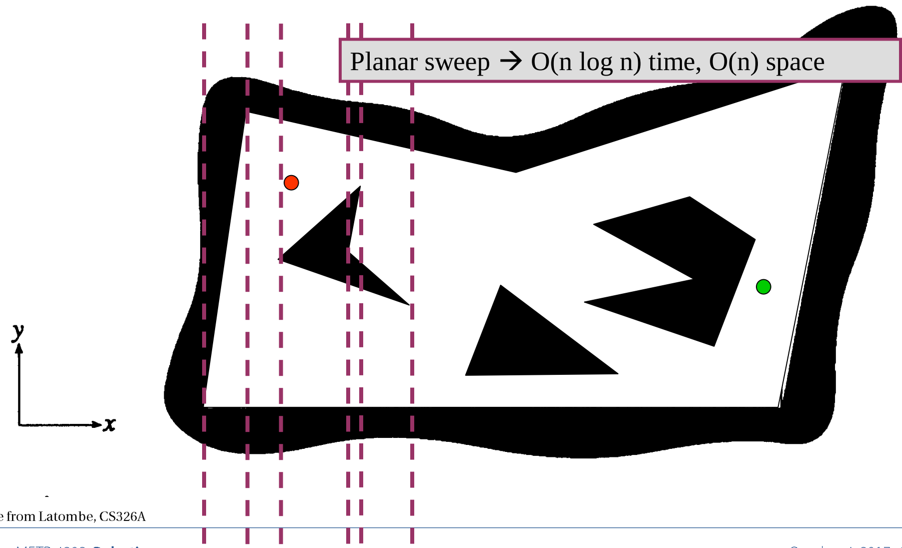
- Exact cell decomposition
  - The free space **F** is represented by a collection of non-overlapping cells whose union is exactly **F**
  - Example: trapezoidal decomposition
- Approximate cell decomposition
  - **F** is represented by a collection of non-overlapping cells whose union is contained in **F** Examples: quadtree, octree, 2n-tree

Trapezoidal decomposition



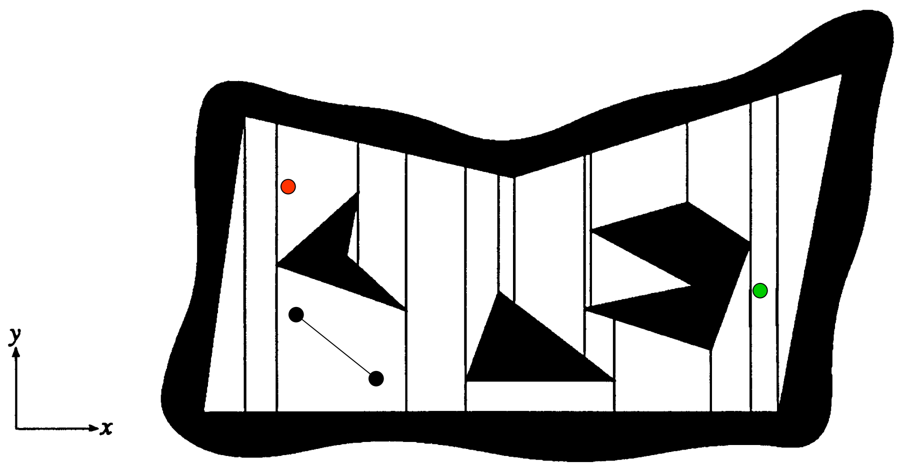
Slide from Latombe, CS326A

Trapezoidal decomposition



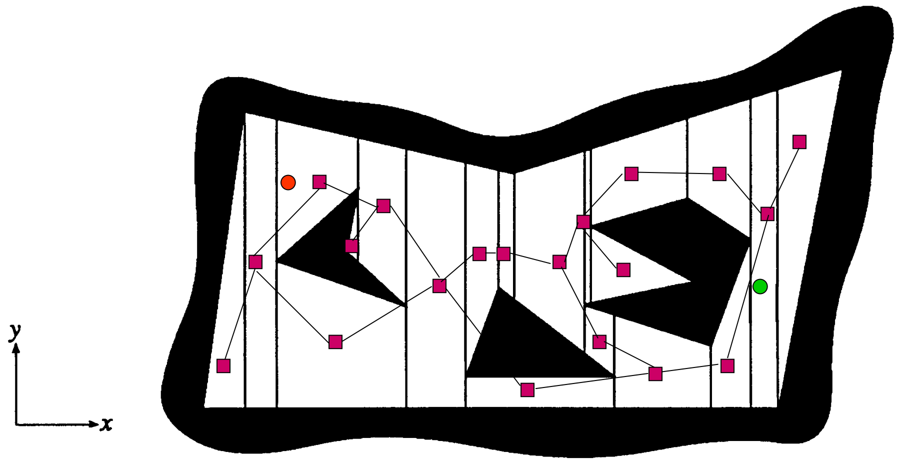
Slide from Latombe, CS326A

Trapezoidal decomposition



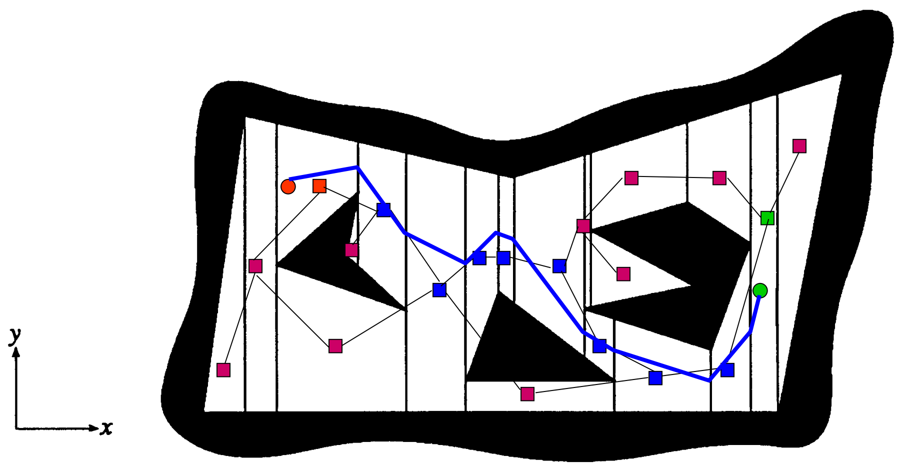
Slide from Latombe, CS326A

Trapezoidal decomposition



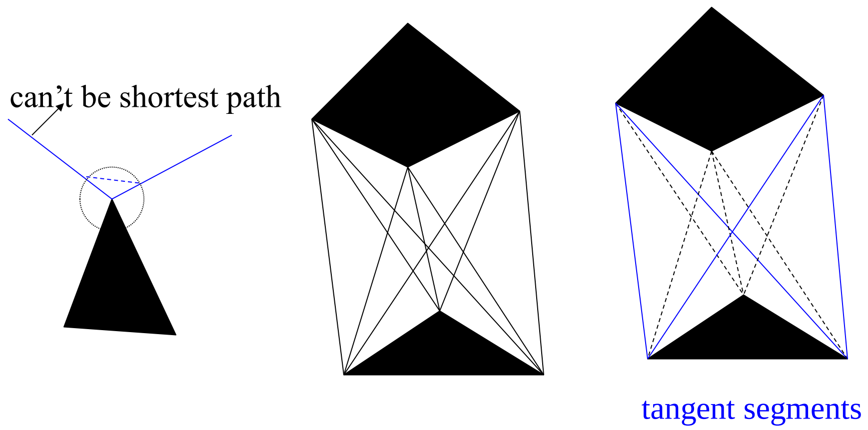
Slide from Latombe, CS326A

Trapezoidal decomposition



Slide from Latombe, CS326A

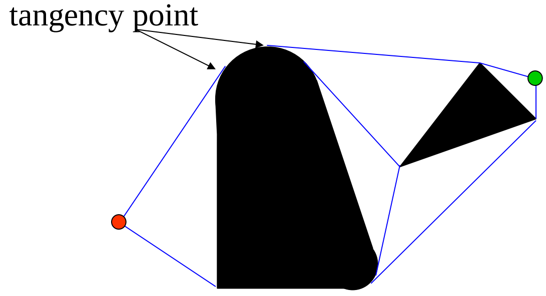
II.Visibility Graph



→ Eliminate concave obstacle vertices

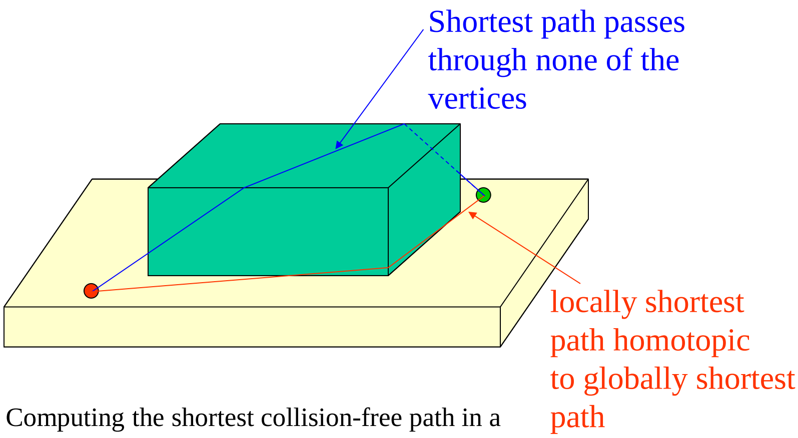
Slide from Latombe, CS326A

Generalized (Reduced) -- Visibility Graph



Slide from Latombe, CS326A

Three-Dimensional Space

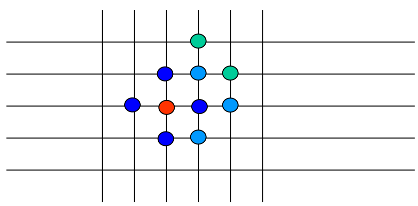


Computing the shortest collision-free path in a polyhedral space is NP-hard

Slide from Latombe, CS326A

Sketch of Grid Algorithm (with best-first search)

- Place regular grid G over space
- Search G using best-first search algorithm with potential as heuristic function



Slide from Latombe, CS326A

Simple Algorithm (for Visibility Graphs)

- Install all obstacles vertices in VG, plus the start and goal positions
- For every pair of nodes u, v in VG

  If segment(u,v) is an obstacle edge then

  insert (u,v) into VG

  else

  for every obstacle edge e

  if segment(u,v) intersects e

  then go up to segment

  insert (u,v) into VG
- Search VG using A*

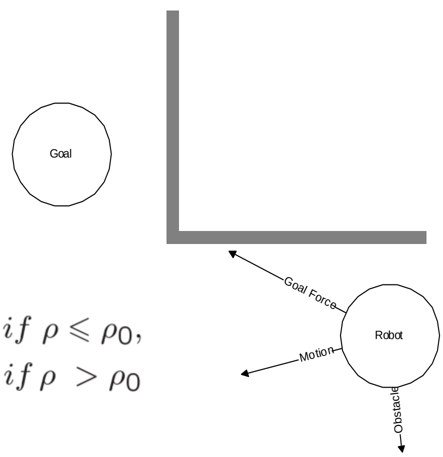
Slide based on Latombe, CS326A

III. Potential Field Methods

- Approach initially proposed for real-time collision avoidance [Khatib, 86]

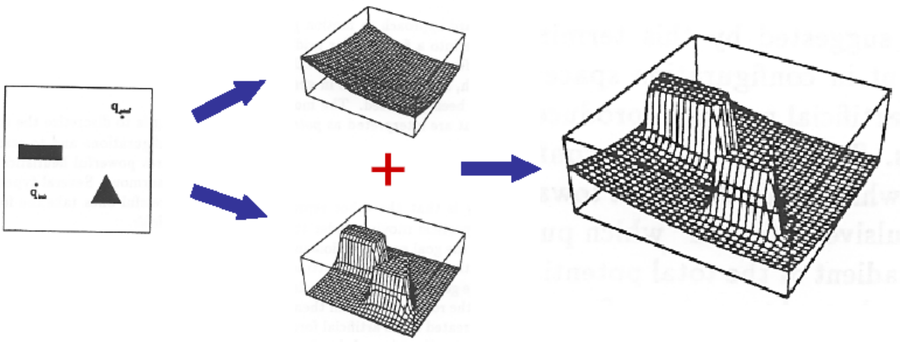
$$F_{Goal} = -k_p(x - x_{Goal})$$

$$F_{Obstacle} = \begin{cases} \eta \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right) \frac{1}{\rho^2}\frac{\partial \rho}{\partial x} & if\ \rho \leqslant \rho_0, \\ 0 & if\ \rho > \rho_0 \end{cases}$$



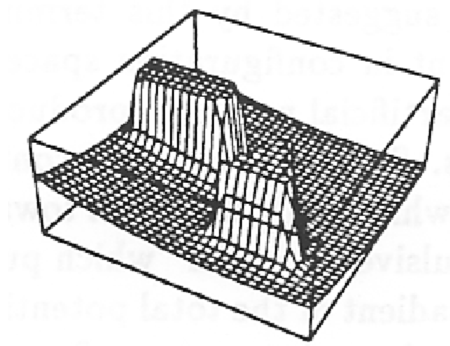
Slide based on Latombe, CS326A

Attractive and Repulsive fields



Slide from Latombe, CS326A

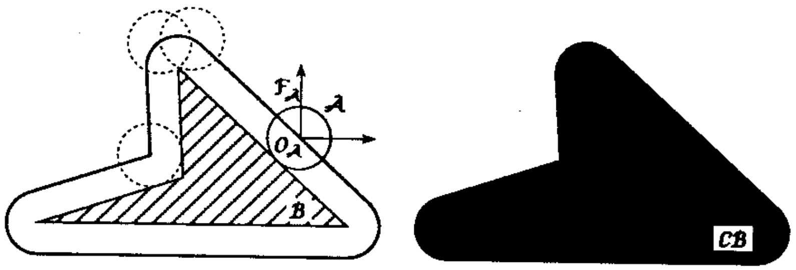
Local-Minimum Issue



- Perform best-first search (possibility of combining with approximate cell decomposition)
- Alternate descents and random walks
- Use local-minimum-free potential (navigation function)

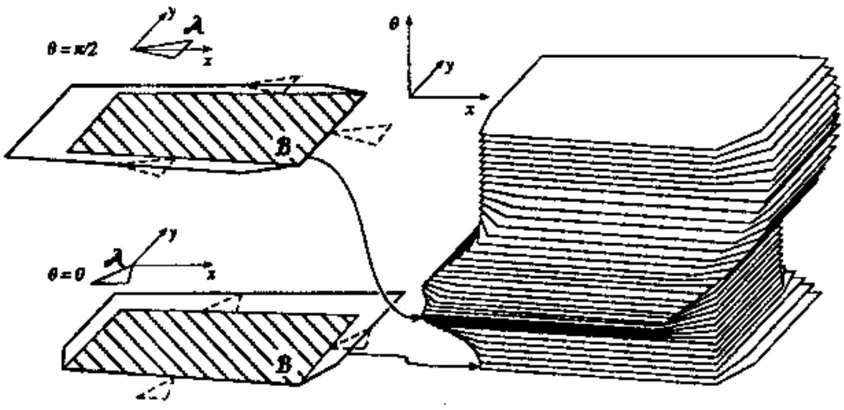
Slide from Latombe, CS326A

Disc Robot in 2-D Workspace



Slide from Latombe, CS326A

Rigid Robot Translating and Rotating in 2-D



Slide from Latombe, CS326A

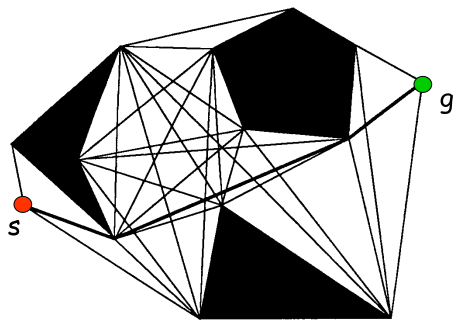
IV. Roadmap Methods

- **Visibility graph**
- **Voronoi diagram**
- Silhouette
  First complete general method that applies to spaces of any dimension and is singly exponential in # of dimensions [Canny, 87]

Slide from Latombe, CS326A

Roadmap Methods

- **Visibility graph**
  Introduced in the Shakey project at SRI in the late 60s. Can produce shortest paths in 2-D configuration spaces

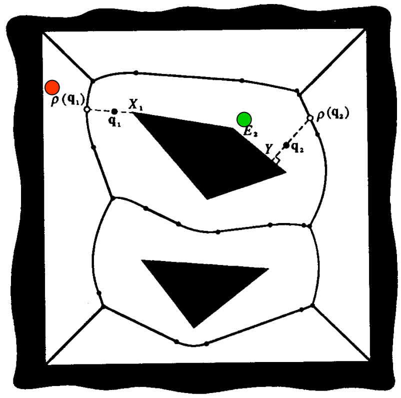


Slide from Latombe, CS326A

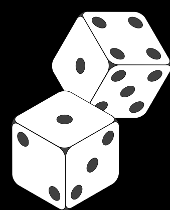
Roadmap Methods

- Voronoi diagram
  Introduced by Computational Geometry researchers. Generate paths that maximizes clearance.

  O(n log n) time
  O(n) space



Slide from Latombe, CS326A



Sample-Based Motion Planning!

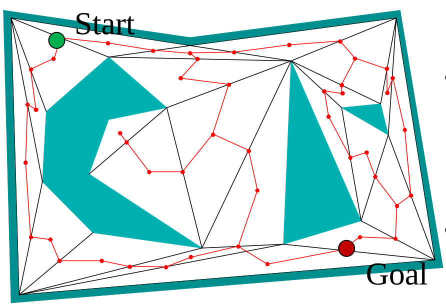
Dice artwork from https://commons.wikimedia.org/wiki/File:2-Dice-Icon.svg

IV. Roadmap Methods

- **Visibility graph**
- **Voronoi diagram**
- Silhouette
  First complete general method that applies to spaces of any dimension and is singly exponential in # of dimensions [Canny, 87]
- **Probabilistic roadmaps (PRMS) and Rapidly-exploring Randomized Trees (RRTs)**

Slide from Latombe, CS326A

Limits of Geometric Planning Methods



- How does this scale to high degrees of freedom?
- What about "dynamic constraints"?
- What about optimality?
- How to tie this to learning and optimization

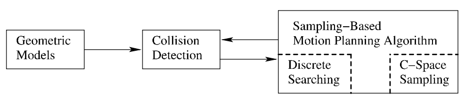
Artwork from LaValle, Ch. 6

But Intel Giveth!

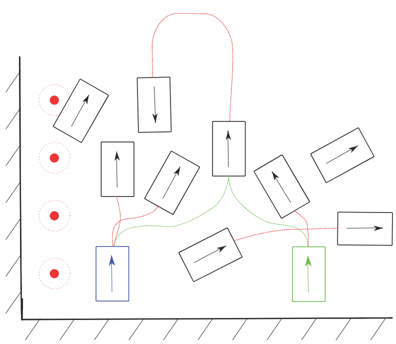
- "Moore's Law" is exponential (at best!)
- These problems ∝ factorial!
- Some Numbers: (From: D. MacKay, Information Theory, Inference, and Learning Algorithms)

	2^{8192}	10^{2466}	Number of distinct 1-kilobyte files
	2^{1024}	10^{308}	Number of states of a 2D Ising model with 32×32 spins
2^{1000}		10^{301}	Number of binary strings of length 1000
2^{500}		3×10^{150}	
	2^{469}	10^{141}	Number of binary strings of length 1000 having 100 1s and 900 0s
	2^{266}	10^{80}	Number of electrons in universe
2^{200}		1.6×10^{60}	
	2^{190}	10^{57}	Number of electrons in solar system
	2^{171}	3×10^{51}	Number of electrons in the earth
2^{100}		10^{30}	
	2^{98}	3×10^{29}	Age of universe/picoseconds
	2^{58}	3×10^{17}	Age of universe/seconds
2^{50}		10^{15}	
2^{40}		10^{12}	
		10^{11}	Number of neurons in human brain
		10^{11}	Number of bits stored on a DVD
		3×10^{10}	Number of bits in the wheat genome
		6×10^{9}	Number of bits in the human genome

Sample-Based Motion Planning



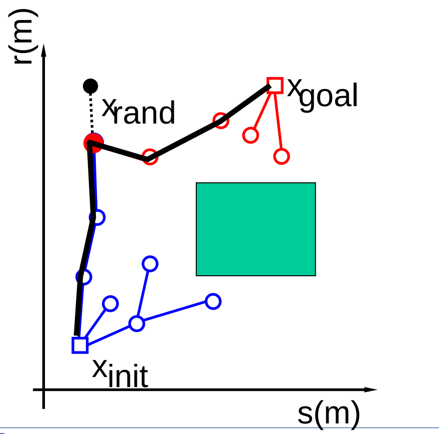
- PRMs



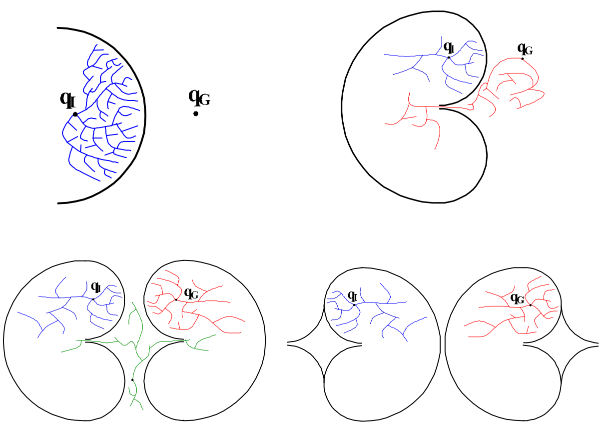
- RRTs

Artwork based on LaValle, Ch. 5

Rapidly Exploring Random Trees (RRT)



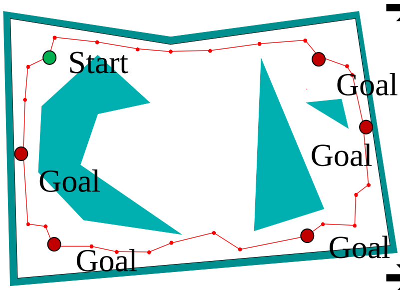
Sampling and the "Bug Trap" Problem



Artwork based on LaValle, Ch. 5

Multiple Points & Sequencing

- Sequencing
  - Determining the “best” order to go in



➔ Travelling Salesman Problem

A salesman has to visit each city on a given list exactly once. In doing this, he **starts** from his home city and in the **end he has to return to his home** city. It is plausible for him to select the order in which he visits the cities so that the **total of the distances travelled** in his tour is as small as possible.

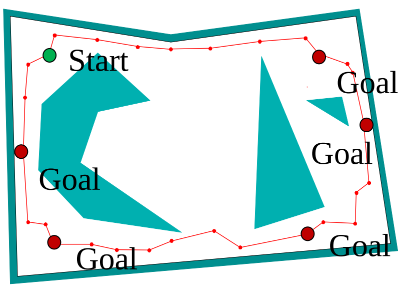
➔ Multi-Goal Problem

A salesman has to visit each city on a given list exactly once. In doing this, he **starts** from his home city and in the **end he has to return to his home** city. It is plausible for him to select the order in which he visits the cities so that the **total of the distances travelled** in his tour is as small as possible.

Artwork based on LaValle, Ch. 6

Travelling Salesman Problem

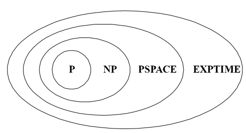
- Given a $n \times n$ distance matrix $\mathbf{C}=(c_{ij})$



- Minimize:

$$c(\pi) = \sum_{i=1}^{n} c_{i\pi(i)}$$

- Note that this problem is NP-Hard

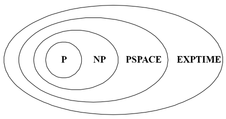


➔ BUT, Special Cases are Well-Solvable!

Artwork based on LaValle, Ch. 6

Travelling Salesman Problem [2]

- This problem is NP-Hard



➔ BUT,
Special Cases are
Well-Solvable!

For the Euclidean case
(where the points are on the 2D Euclidean plane) :

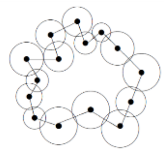
- The shortest TSP tour does not intersect itself, and thus geometry makes the problem somewhat easier.
- If all cities lie on the boundary of a convex polygon, the optimal tour is a cyclic walk along the boundary of the polygon (in clockwise or counterclockwise direction).

***The k*-line TSP**

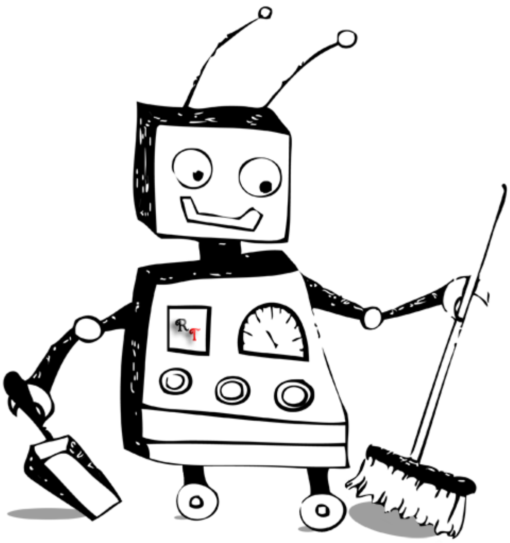
- The a special case where the cities lie on *k* parallel (or almost parallel) lines in the Euclidean plane.
- EG: Fabrication of printed circuit boards
- Solvable in **O**(n^3) time by Dynamic Programming (Rote's algorithm)

The necklace TSP

- The special Euclidean TSP case where there exist *n* circles around the *n* cities such that every cycle intersects exactly two adjacent circles



Cool Robotics Share



Optional:
Search Refreshers!

From: Tuomas Sandholm (CSD, CMU) and N. H. Reyes (Massey [NZ])

“Heuristic”

- Literally translates to “to find”, “to discover”
- Has many meanings in general
  - How to come up with mathematical proofs
  - Opposite of algorithmic
  - Rules of thumb in expert systems
  - Improve average case performance, e.g., in CSPs
  - Algorithms that use low-order polynomial time (and come within a bound of the optimal solution)
    - % from optimum
    - % of cases
    - “probably approximately correct”
  - h(n) that estimates the remaining cost from a state n to a solution
    - We’ll assume that for all n, h(n) ≥ 0, and for all goal nodes n, h(n)=0.

Uninformed vs. Informed Search

Uninformed

Can only generate successors and distinguish goals from non-goals

Informed

Strategies that know whether one non-goal is more promising than another

Best-First Search

f(n)

function BEST-FIRST-SEARCH (*problem*, EVAL-FN) **returns** a solution sequence
inputs: *problem*, a problem
Eval-Fn, an evaluation function

Queuing-Fn – a function that orders nodes by EVAL-FN
return GENERAL-SEARCH (*problem*, *Queuing-Fn)*

An implementation of best-first search using the general search algorithm.

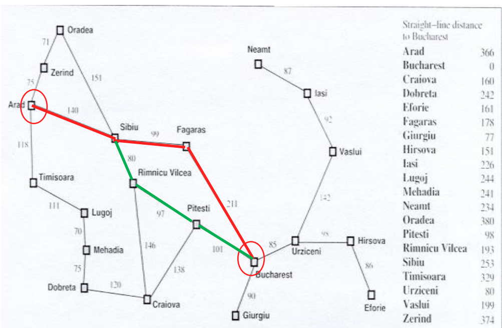
Usually, knowledge of the problem is incorporated in an evaluation function that describes the desirability of expanding the particular node.

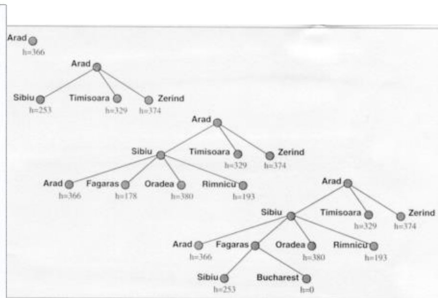
If we really knew the desirability, it would not be a search at all. So, we should really call it "seemingly best-first search" to be pedantic

Greedy Search

function GREEDY-SEARCH (*problem*) **returns** a solution or failure
return BEST-FIRST-SEARCH (*problem*, *h)*

h(n) = estimated cost of the cheapest path from the state at node n to a goal state





Not Optimal
Incomplete
O(b^m) time
O(b^m) space

Beam Search

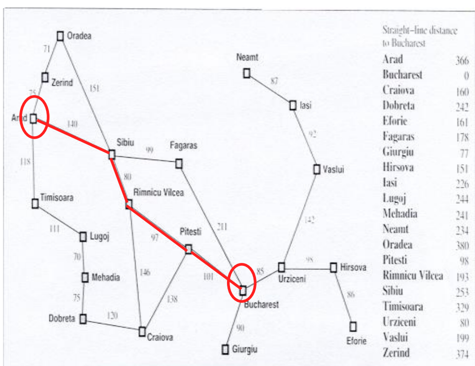
Use f(n) = h(n) but |nodes| ≤ K

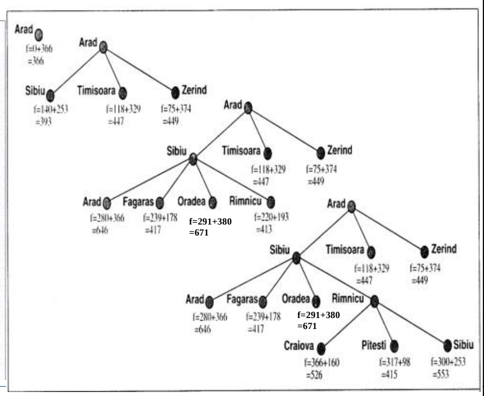
- Not complete
- Not optimal

A* Search

function A*-SEARCH (*problem*) **returns** a solution or failure
  return BEST-FIRST-SEARCH (*problem*, *g+h*)

f(n) = estimated cost of the cheapest solution through n
  = g(n) + h(n)





A* Search…

In a minimization problem, an admissible heuristic h(n) *never overestimates the real value*
(In a maximization problem, h(n) is admissible if it never *under*estimates)

Best-first search using f(n) = g(n) + h(n) and an admissible h(n) is known as *A* search*

A* tree search is complete & optimal

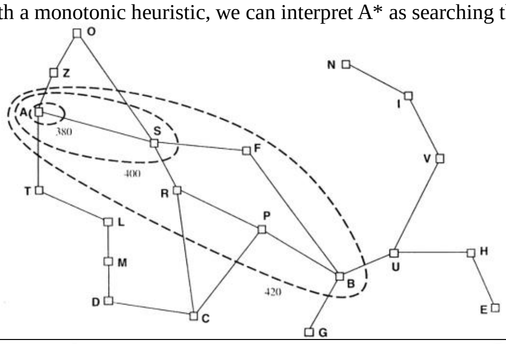
Completeness of A*

- Because A* expands nodes in order of increasing f, it must eventually expand to reach a goal state. This is true unless there are infinitely many nodes with $f(n) \leq f^*$
- How could this happen?
  - There is a node with an infinity branching factor
  - There is a path with finite path cost but an infinite number of nodes on it
- So, A* is complete on graphs with a finite branching factor provided there is some positive constant δ such that every operator costs at least δ

Monotonicity of a heuristic

- h(n) is monotonic (aka. consistent) if, for every node n and every child n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n': $h(n) \leq c(n,a,n') + h(n')$.
- Monotonicity implies that f(n) (which equals g(n)+h(n)) never decreases along a path from the root.
- Monotonic => admissible
- Advanced topic: "A* Search with Inconsistent Heuristics" in Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, provides techniques for making non-monotonic heuristics monotonic

With a monotonic heuristic, we can interpret A* as searching through contours:



Map of Romania showing contours at f = 380, f = 400 and f = 420, with Arad as the start state. Nodes inside a given contour have f-costs lower than the contour value.

Monotonicity of a heuristic…

- A* expands all nodes n with f(n) < f*, and may expand some nodes right on the "goal contour" (f(n) = f*), before selecting a goal node.
- With a monotonic heuristic, even A* *graph* search (i.e., search that deletes later-created duplicates) is optimal.
  - Another option, which requires only admissibility – not monotonicity – is to have the duplicate detector always keep the best (rather than the first) of the duplicates.
- A* *tree* search doesn't do detection or removal of duplicates

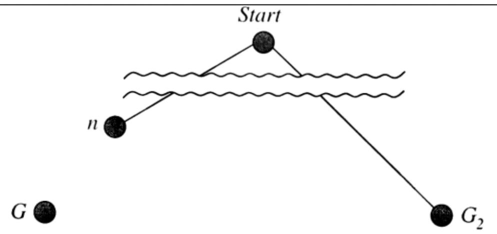
Proof of optimality of A* tree search

Assumes h is admissible, but does not assume h is monotonic

Let G be an optimal goal state, and $f(G) = f^* = g(G)$.
Let G_2 be a suboptimal goal state, i.e., $f(G_2) = g(G_2) > f^*$.
Suppose for contradiction that A* has selected G_2 from the queue. (This would terminate A* with a suboptimal solution.)

Let n be a node that is currently a leaf node on an optimal path to G.

Situation at the point where a sub-optimal goal state G_2 is about to be picked from the queue



Because h is admissible, $f^* \geq f(n)$.

If n is not chosen for expansion over G_2, we must have $f(n) \geq f(G_2)$.
So, $f^* \geq f(G_2)$. Because $h(G_2)=0$, we have $f^* \geq g(G_2)$, contradiction. □

Complexity of A*

- Generally $O(b^d)$ time and space.
- Sub-exponential growth when $|h(n) - h^*(n)| \leq O(\log h^*(n))$
    - Unfortunately, for most practical heuristics, the error is at least proportional to the path cost

A* is optimally efficient

- A* with a monotonic heuristic is optimally efficient for any given h-function among algorithms that extend search paths from the root. I.e., no other optimal algorithm is guaranteed to expand fewer nodes (except perhaps on the goal contour where f(n)=f*) (for a given search formulation).
  - With a non-monotonic admissible heuristic, some nodes can be expanded many times, causing the search to do $O(2^N)$ node expansions, where N is the number of nodes expanded

- Intuition: any algorithm that does not expand all nodes in the contours between the root and the goal contour runs the risk of missing the optimal solution.

Heuristics (h(n)) for A*

A typical instance of the 8-puzzle

5	4	
6	1	8
7	3	2

Start state

1	2	3
8		4
7	6	5

Goal state

Heuristics?

h_1: #tiles in wrong position

h_2: sum of Manhattan distances of the tiles from their goal positions

h_2 dominates h_1: $\forall n,\ h_2(n) \geq h_1(n)$

Heuristics (h(n)) for A* …

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Comparison of the search costs and effective branching factors for the ITERATIVE-DEPENING-SEARCH and A* algorithms with *h*1, *h*2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

Always, if a heuristic h_2 dominates h_1, then A* using h_2 will never expand more nodes than A* using h_1 (except possibly for some nodes with f(n)=f*).
<= A* expands all nodes with f(n) < f*

Inventing heuristic functions h(n)

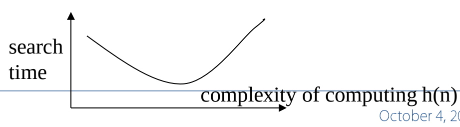
Cost of exact solution to a relaxed problem is often a good heuristic for original problem.
Relaxed problem(s) can be generated automatically from the problem description by dropping or relaxing constraints.
Most common example in operations research: relaxing all integrality constraints and using linear programming to get an optimistic h-value.

What if no dominant heuristic is found?
$h(n) = \max [h_1(n), \ldots h_m(n)]$
h(n) is still admissible & dominates the component heuristics

Use probabilistic info from statistical experiments: "If h(n)=14, h*(n)=18".
Gives up optimality, but does less search

Pick features & use machine learning to determine their contribution to h.

Use full breath-first search as a heuristic?



Memory-bounded search algorithms

Iterative Deepening A* (IDA*)

```
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f-COST limit
          root, a node

  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f-COST(root)
  loop do
      solution, f-limit ← DFS-CONTOUR(root,f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end

function DFS-CONTOUR(node,f-limit) returns a solution sequence and a new f-COST limit
  inputs: node, a node
          f-limit, the current f-COST limit
  static: next-f, the f-COST limit for the next contour, initially ∞

  if f-COST[node] > f-limit then return null, f-COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSOR(node) do
     solution, new-f ← DFS-CONTOUR(s,f-limit)
     if solution is non-null then return solution, f-limit
     next-f ← MIN(next-f, new-f); end
  return null, next-f
```

f-COST[node] = g[node] + h[node]

IDA* …

Complete & optimal under same conditions as A*.
Linear space. Same O( ) time complexity as A*.

If #nodes grows exponentially, then asymptotically optimal space.

IDA* …

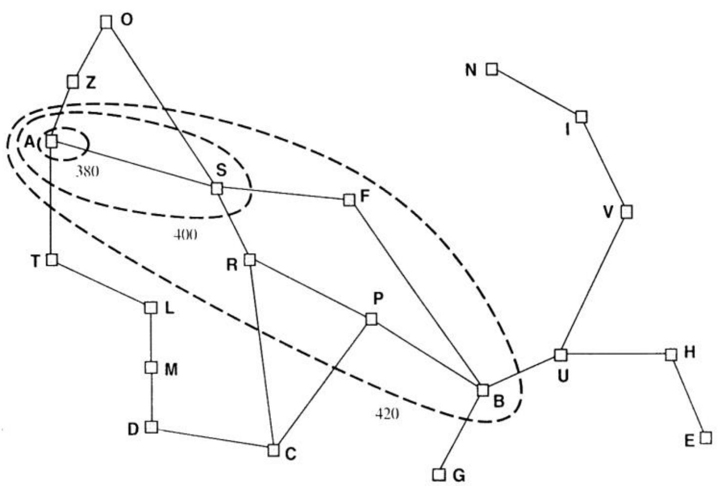
Effective e.g. in 8-puzzle where f typically only increases 2-3 times → 2-3 iterations.

Last iteration ~ A*

Ineffective in, e.g., TSP where f increases continuously
→ each new iteration only includes one new node.

- If A* expands N nodes, IDA* expands $O(N^2)$ nodes
- Fixed increment ε → ~1/ε iterations
- Obtains ε-optimal solution if terminated once first solution is found
- Obtains an optimal solution if search of the current contour is completed

A* vs. IDA*



Map of Romania showing contours at f = 380, f = 400 and f = 420, with Arad as the start sate. Nodes inside a given contour have f-costs lower than the contour value.

Memory-bounded search algorithms

- IDA* 1985
- Recursive best-first search 1991
- Memory-bounded A* (MA*) 1989
- Simple memory-bounded A* (SMA*) 1992 …

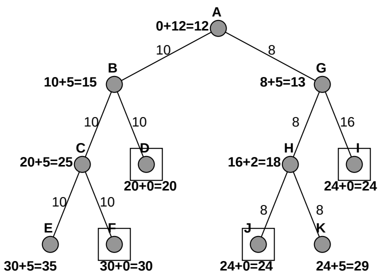
} use "too little" memory

Simple Memory-bounded A* (SMA*)

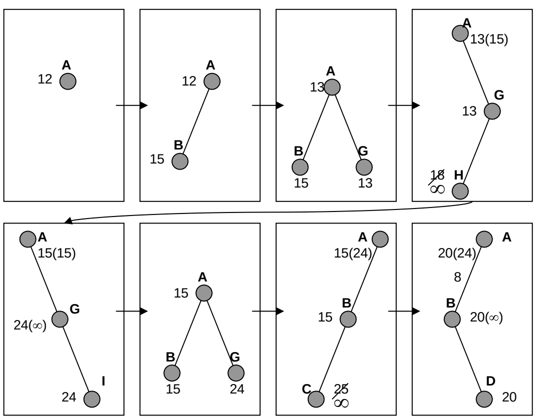
Search *space*

f = g+h

□ = goal



Progress of SMA* (with enough memory to store just 3 nodes).
Each node is labeled with its *current* *f*-cost.
Values in parentheses show the value of the best forgotten descendant.



Optimal & complete if enough memory
Can be made to signal when the best solution found might not be optimal (e.g., if J=19)

D* Motion Planner (Dynamic A* **RePlanner**)

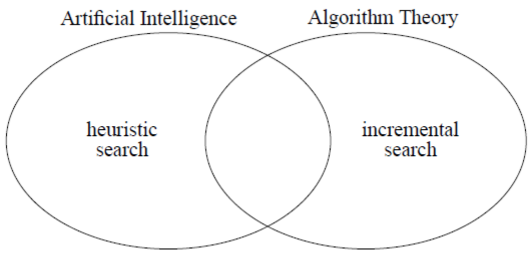
D* Lite

- an incremental version of A*
- for navigating in unknown terrain

It implements the same behavior as Stentz' **Focussed Dynamic A*** but is algorithmically different.

n.h.reyes@massey.ac.nz

Incremental search + heuristic search

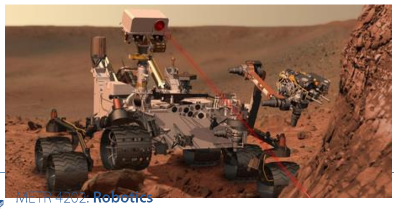


How to search efficiently using heuristic to guide the search

How to search efficiently by using re-using information from previous search results

Focussed Dynamic A* (D*)

- Stentz 1995
- Clever heuristic method that achieves a speedup of one to two orders of magnitudes over repeated **A*** searches
- The improvement is achieved by modifying previous search results locally
- Extensively used on real robots, including outdoor high mobility multi-wheeled vehicle (HMMWV)
- Integrated into Mars Rover prototypes and tactical mobile robot prototypes for urban reconnaissance



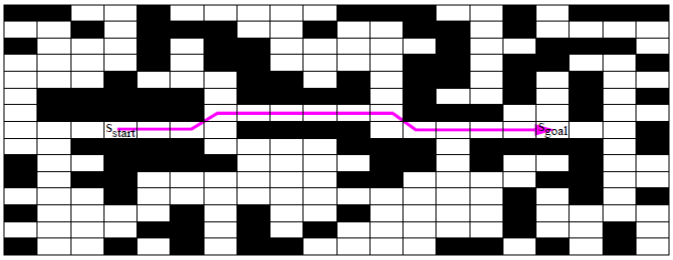
D* Lite vs. D*

- **D* Lite** implements the same navigation strategy as **D***, but is algorithmically different
- Substantially shorter than **D***
- Uses only one tie-breaking criterion when comparing priorities (simplified maintenance)
- No nested if statements with complex conditions
- Simplifies the analysis of program flow
- Easier to extend
- At least as efficient as **D***

Previously, we learned LPA*

LPA* repeatedly determines shortest paths between S_{start} and S_{goal} as the edge costs of a graph change.

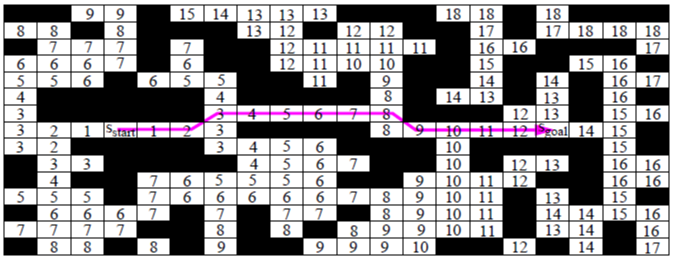
original eight-connected gridworld



Path Planning

LPA* repeatedly determines shortest paths between **S$_{start}$** and **S$_{goal}$** as the edge costs of a graph change.

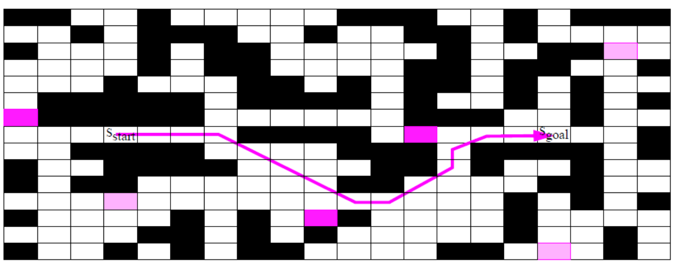
original eight-connected gridworld



Path Planning

LPA* repeatedly determines shortest paths between **S$_{start}$** and **S$_{goal}$** as the edge costs of a graph change.

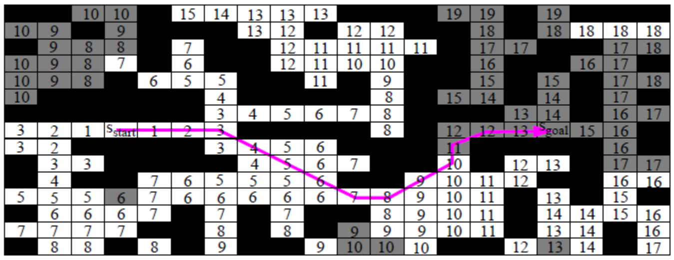
changed eight-connected gridworld



Path Planning

LPA* repeatedly determines shortest paths between **S_{start}** and **S_{goal}** as the edge costs of a graph change.

changed eight-connected gridworld



LPA*

- LPA* is an incremental version of A* that applies to the same finite path-planning problems as A*.
- It shares with A* the fact that it uses non-negative and consistent heuristics h(s) that approximate the goal distances of the vertices s to focus its search.
- Consistent heuristics obey the triangle inequality:
  - h(sgoal) = 0
  - h(s) $\leq$ c(s, s’) + h(s’); for all vertices s ∈ S and
  - s’ ∈ succ(s) with s ≠ sgoal.

D* Lite

LPA* repeatedly determines shortest paths between S_{start} and S_{goal} as the edge costs of a graph change.

D* Lite repeatedly determines shortest paths between the current vertex $S_{current}$ of the robot and S_{goal} as the edge costs of a graph change, while the robot moves towards S_{goal}.

D* Lite

LPA* repeatedly determines shortest paths between S_{start} and S_{goal} as the edge costs of a graph change.

D* Lite repeatedly determines shortest paths between the current vertex $S_{current}$ of the robot and S_{goal} as the edge costs of a graph change, while the robot moves towards S_{goal}.

D* Lite is suitable for solving **goal-directed navigation problems** in **unknown terrains**.

Autonomy Has Limits