Motion Planning

Advanced Search-Based Motion Planning

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

May 15, 2024

Recap: A*

Input

- Weighted graph
  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ $d : \mathcal{E} \to \mathbb{R}$
- Start $v_s \in \mathcal{V}$, Goal $v_g \in \mathcal{V}$
- *admissible* heuristic $h : \mathcal{V} \to \mathbb{R}$

Output

optimal path in the graph; formally:

$(v_1, v_2, \dots, v_n)$ $v_i \in \mathcal{V}$ $(v_i, v_{i+1}) \in \mathcal{E}$

$v_1 = v_s$ $v_n = v_g$

$\min \sum_{i=1}^{n-1} d((v_i, v_{i+1}))$

Properties

- complete
- optimal / admissible
- optimal efficient

Insight

Expansion using priority queue ordered by

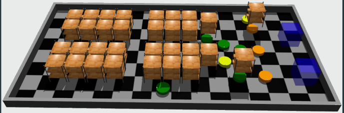
$f(v) = g(v) + h(v)$

Applications

- Route planner
- Robot navigation
- Computer games

Challenges for Robot Motion Planning

Realtime operation in dynamic environments



Need to be able to react to unforeseen changes quickly.

Non-Holonomic Robots

Source: New Venturist

Some robots cannot move in a 4-connected grid.

Large Robots

Source: Örebro University

Grid-size needs to be at least as large as the robot.

Important A* Variants

Bounded Suboptimal Planning

- A* finds a solution and a proof of its optimality
- Sometimes the optimal solution is not needed

Bounded suboptimal planning

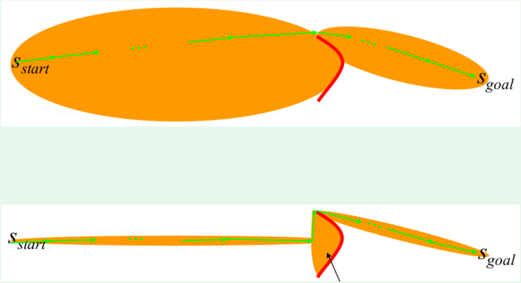
A solution is ϵ-optimal if its cost c is at most a factor of ϵ larger than the optimal cost c^*:

$$c \leq \epsilon c^*.$$

Bounded Suboptimal Planning: Weighted-A* (wA*) [2]

- A*: Expansion order
  $f(v) = g(v) + h(v)$
- Weighted-A*: Expansion order
  $f(v) = g(v) + \epsilon h(v)$, where $\epsilon \geq 1$
- $\epsilon > 1$: Bias towards states that seem to be closer to goal
- Weighted-A* is ϵ-optimal and works well in practice [1]

State expansions A* vs. Weighted-A*

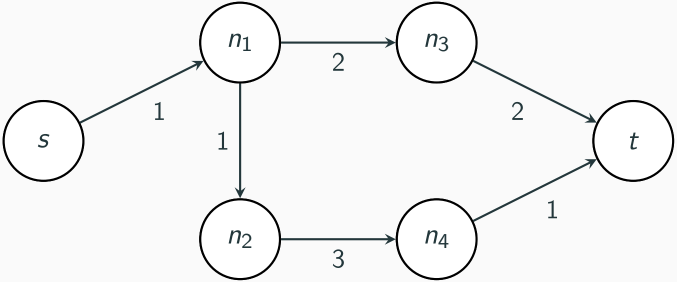


Bounded Suboptimal Planning: A^*_ϵ (Focal Search) [3]

- FOCAL: subset of OPEN with $f(v)$ at most $\epsilon f(top(OPEN))$ (up to a factor ϵ larger than the smallest value in OPEN)
- FOCAL is a priority queue, sorted by an arbitrary heuristic function
- Search is identical to A*, but takes smallest element from FOCAL
- Focal search is ϵ-optimal

Example, where an inadmissible heuristic can be used!

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



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

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

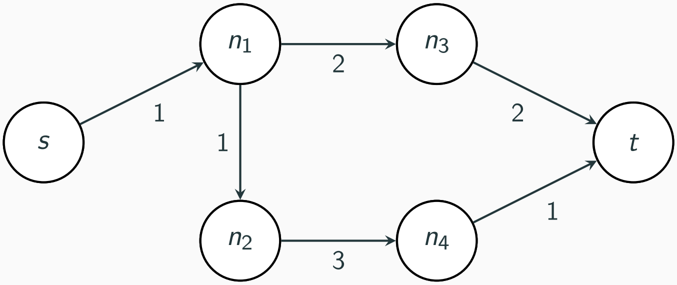
$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$

OPEN $= \langle\rangle$

FOCAL $= \langle\rangle$

Next state to expand:

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



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

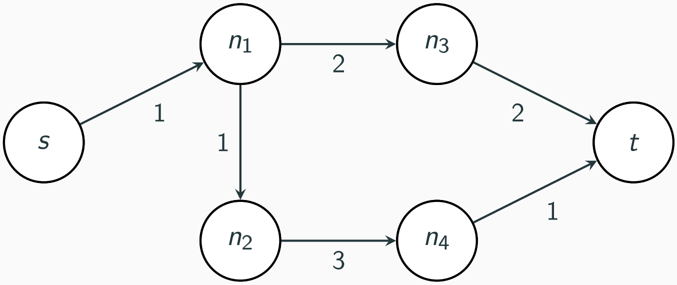
$h(s) = 3, h(n_1) = 2, h(n_2) = 2, h(n_3) = 1, h(n_4) = 1, h(t) = 0$

$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$

OPEN $= \langle \mathbf{s} \rangle$

FOCAL $= \langle s \rangle$ Next state to expand: s

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



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

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

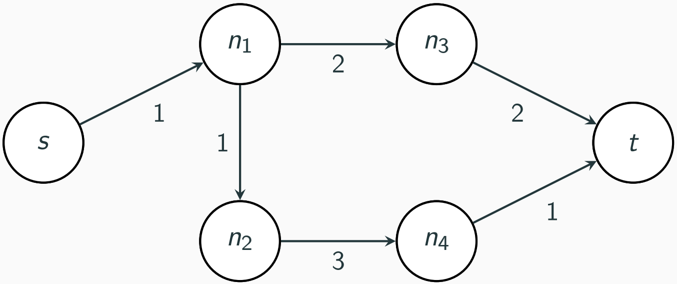
$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$

$\text{OPEN} = \langle n_1 \rangle$

$\text{FOCAL} = \langle n_1 \rangle$

Next state to expand: n_1

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



$g(s) = 0, g(n_1) = 1, \mathbf{g(n_2) = 2}, \mathbf{g(n_3) = 3}, g(n_4) = \infty, g(t) = \infty$

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

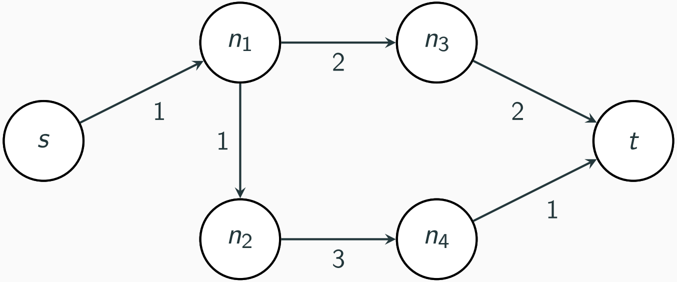
$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$

OPEN $= \langle n_2, n_3 \rangle$

FOCAL $= \langle n_3, n_2 \rangle$

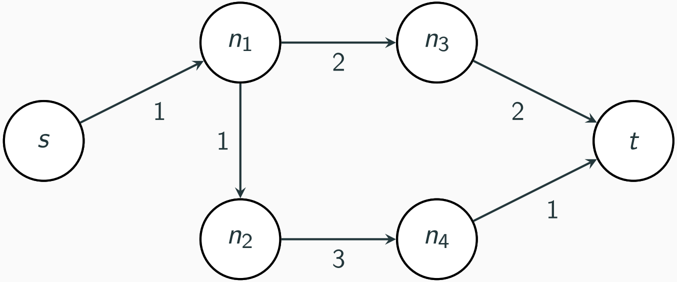
Next state to expand: n_3

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



$g(s) = 0, g(n_1) = 1, g(n_2) = 2, g(n_3) = 3, g(n_4) = \infty, \mathbf{g(t) = 5}$
$h(s) = 3, h(n_1) = 2, h(n_2) = 2, h(n_3) = 1, h(n_4) = 1, h(t) = 0$
$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$
$\text{OPEN} = \langle n_2, t \rangle$
$\text{FOCAL} = \langle t, n_2 \rangle$ Next state to expand: t (Assuming $\epsilon > \frac{5}{4}$)

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



$g(s) = 0, g(n_1) = 1, g(n_2) = 2, g(n_3) = 3, g(n_4) = \infty, g(t) = 5$

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

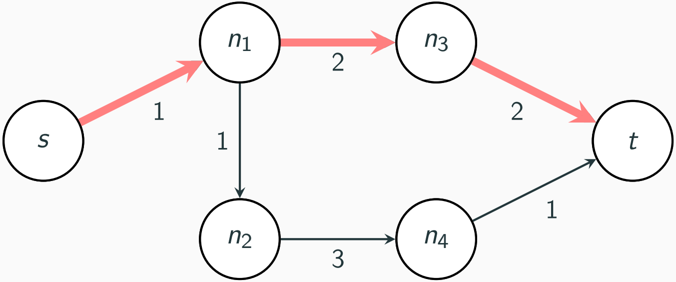
$h_2(s) = 0, h_2(n_1) = 10, h_2(n_2) = 100, h_2(n_3) = 1, h_2(n_4) = 2, h_2(t) = 0$

$\text{OPEN} = \langle n_2 \rangle$

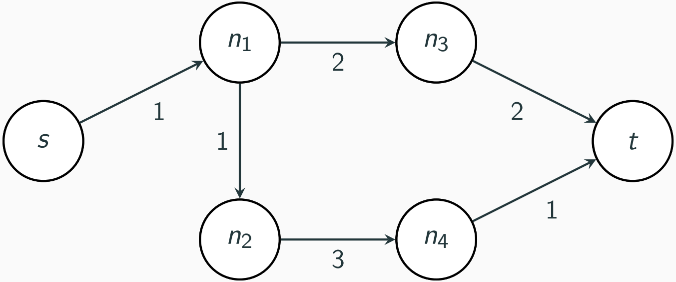
$\text{FOCAL} = \langle n_2 \rangle$

Next state to expand:

Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



Bounded Suboptimal Planning: A^*_ϵ (Focal Search)



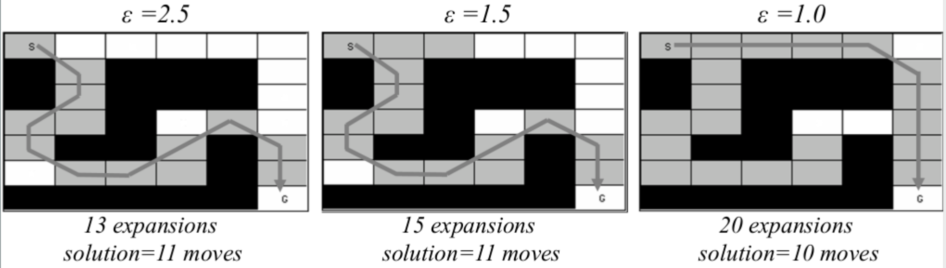
What would change if $\epsilon \leq \frac{5}{4}$?

Anytime Planning

- Need to be able to (re-)plan in realtime
- Anytime planning: return best possible solution for a given time limit

Approach 0: Series of suboptimal planners

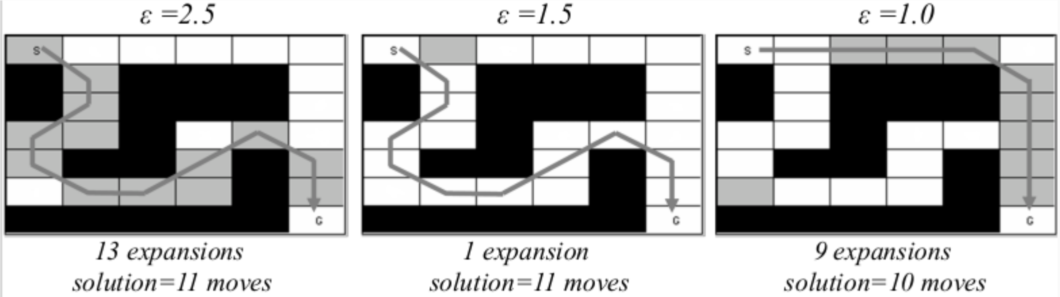
Execute a sequence of bounded suboptimal planners with decreasing ϵ



Main drawbacks: Inefficient (many states identical between iterations)

Anytime Planning: Anytime Repairing A* (ARA*) [4]

- Add $g_e(v)$ for each vertex v: when expanding v, we set $g_e(v) = g(v)$
- ARA* keeps track of $g_e(v)$ between iterations
- Initializes OPEN with states that fulfill $g_e(v) > g(v)$
- Execute search weighted-A* only until $f(v_g) > f(top(OPEN))$



Incremental Planning

- Need to be able to react to changes in the graph
- Incremental planning: re-use previous search results by interleaving planning and execution

D* Lite [5]

- Keep track of $g_e(v)$ between iterations
- Special handling for the case where edge cost increases ($g_e(v) < g(v)$)

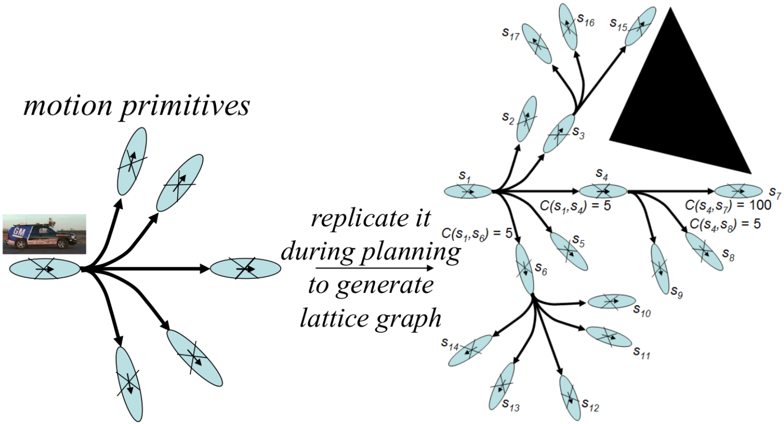
Anytime D* [6]

- Mix of ARA* and D* Lite
- Interleave anytime planning and execution

State Lattices

Key Idea of State Lattices

- Graph $\{V, E\}$ where
  - V: centers of the grid-cells
  - E: motion primitives that connect centers of cells via short-term **feasible** motions



Source: Maxim Likhachev

How Does This Affect A*?

```
def Astar(G, d, v_s, v_g, h):
  O = queue(v_s)
  while O ≠ ∅:
    # smallest f-value
    v = O.pop()
    if v = v_g:
      return solution
    for n in v.neighbor:
      g = v.g + d(v, n)
      if g < n.g:
        O.add_or_update(n, g + h(n))
  return None
```

Discussion

What changes are needed to use state lattices?

Motion Primitive

- A valid trajectory that obeys robot dynamics

This is similar to the "kinodynamic motion planning" definition, without the cost function and free workspace constraint!

In practice, we often like each primitive to be (near)-optimal with respect to some J.

Motion Primitive

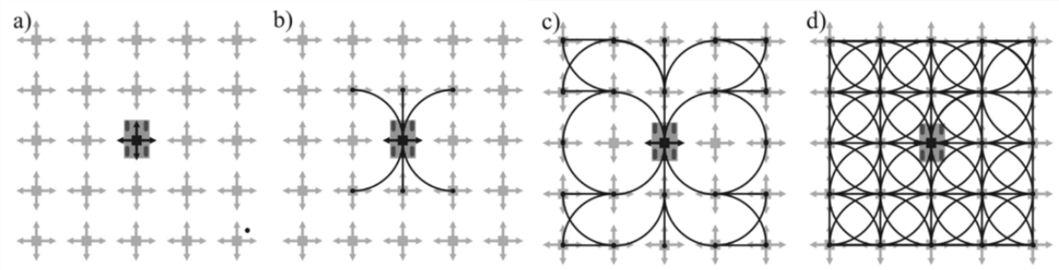
A tuple $\langle T, \mathbf{u}(t), \mathbf{q}(t) \rangle$, where T is the duration, $\mathbf{u} : [0, T) \rightarrow \mathcal{U}$ is the sequence of controls, and $\mathbf{q} : [0, T] \rightarrow \mathcal{Q}$ the sequence of configurations such that:

$$\dot{\mathbf{q}}(t) = \mathbf{f}(\mathbf{q}(t), \mathbf{u}(t)) \quad \forall t \in [0, T).$$

Thus, a motion primitive is specific to a given robot type specified by the robot dynamics $\mathbf{f}$.

State Lattice (1)

- Discretize the configuration space $\mathcal{Q}$ using a lattice
- Example: Car, where x, y, θ is discretized



Source: [7]

State Lattice (2)

State Lattice Motion Primitive

A motion primitive with the additional constraints:

$$\mathbf{q}(0) \in \mathcal{Q}_d$$
$$\mathbf{q}(T) \in \mathcal{Q}_d,$$

where $\mathcal{Q}_d \subset \mathcal{Q}$ is the discretized configuration space.

Change 1: Neighbor Computation

```
def Astar(G, d, v_s, v_g, h):
  O = queue(v_s)
  while O ≠ ∅:
    # smallest f-value
    v = O.pop()
    if v = v_g:
      return solution
    for n in v.neighbor:
      g = v.g + d(v, n)
      if g < n.g:
        O.add_or_update(n, g + h(n))
  return None
```

A neighbor of v is a state lattice motion primitive such that:

- $\mathbf{q}(0) = v$
- The resulting motion is collision-free

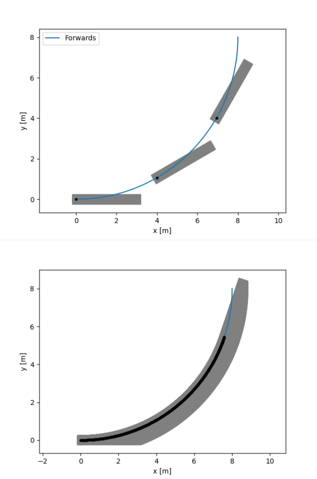
Motion primitives are usually pre-computed. Finding suitable candidates can be done efficiently ($\mathcal{O}(1)$) when using a hash map data structure.

Change 2: Validity Checking: Option 1

- Compute swept volume that is covered by motion primitive
- Check for intersections with the obstacles
- Known as continuous collision checking in FCL

- Only works for linear motions out-of-the-box
- Swept volume can be difficult to compute
- Slow (requires collision check for each expansion)

+ Accurate

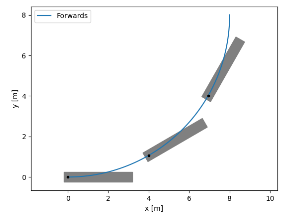


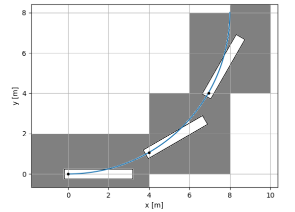
Change 2: Validity Checking: Option 2

- Compute swept grid cells that are covered by motion primitive

- Pessimistic approximation

+ Very efficient





Note: Both approaches can pre-compute the swept volumes

Change 3: Cost and Heuristics (1)

- We need to assume an additive cost function

$$J(T, \mathbf{u}(t), \mathbf{q}(t)) = \sum_{\langle T_i, \mathbf{u}_i(t), \mathbf{q}_i(t) \rangle \in \mathcal{M}} J_i(T_i, \mathbf{u}_i(t), \mathbf{q}_i(t)),$$

  where $\mathcal{M}$ are the primitives used for the solution
- $J_i(\cdot)$ is the cost for a motion primitive (can be precomputed)
- Heuristic needs to be admissible, i.e., never overestimate the true cost

Heuristic for minimal-time-cost

If $J_i(T_i, \mathbf{u}_i(t), \mathbf{q}_i(t)) = T_i$ and we have a first-order model with a maximum speed s_{max}, an admissible heuristic is:

$$h(v) = \frac{\|v - v_g\|_2}{s_{max}}.$$

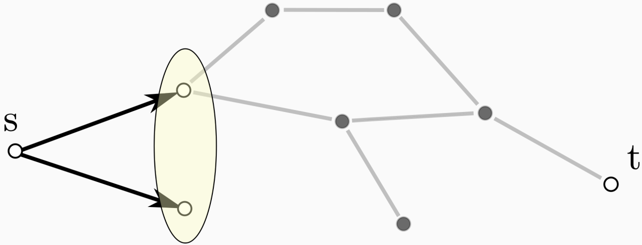
Change 3: Cost and Heuristics (2)

- Heuristic ideally uses strong knowledge of geometry (e.g., true shortest distance) and dynamics
- Can be difficult to define for some cost functions (e.g., minimal-energy)

Change 4: Use Implicit Graph Representation

Implicit graph search

1. Initialize start state s
2. Define successor function Γ
3. Search graph by expanding the next best node



Theoretical Properties

A* is complete, optimal (admissible), and optimal efficient

Discussion

Do these properties hold for planning on state lattices?

- Interpretation 1: Planning on state lattices retains these properties with respect to the discretization (choice of motion primitives)
  - In the limit (infinite many primitives) the properties hold for the continuous problem
- Interpretation 2: Approach is incomplete (limit case not practical)

Practical Challenges: Compute Motion Primitives

- Assume $\mathbf{q}(0)$, $\mathbf{q}(T)$ are selected
- The motion primitive computation, becomes a "small" kinodynamic motion planning problem itself
  - Often termed Two-Value Boundary Problem (TVBP)
  - Can be solved with any other kinodynamic motion planner
  - Frequently used: optimization-based approaches (part 4 of this class) or domain knowledge

Practical Challenges: Select Motion Primitives

- What is a good discretization schema for the lattice?
    - Try to identify symmetries and invariances (e.g., translation-invariance)
    - Domain expert approach: manually select based on robot and environment
    - Data-driven approach: use near-optimal examples to identify recurring motions (e.g., maximum acceleration in a straight line)

Practical Challenges: Scalability

State Lattices scale poorly

The number of primitives grows exponential with the number of states.

Examples and more details in exercise 4!

- Successful application for up to 12-dim state spaces
- Often requires careful tuning and expert knowledge

Case Studies

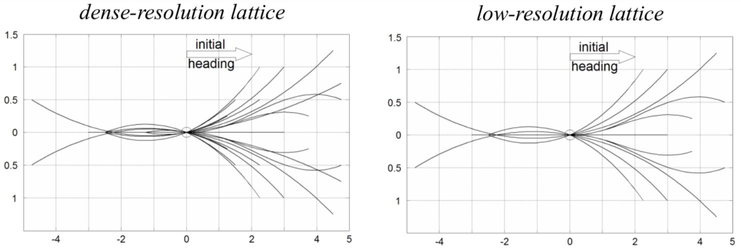
DARPA Urban Challenge (2007) [8]

- Competition for autonomous vehicle to operate in urban environment

- The team that won used a search-based motion planner

DARPA Urban Challenge (2007) [8]

- State space: (x, y, θ, v) (position, orientation, and speed)
- Multi-resolution lattice-graph

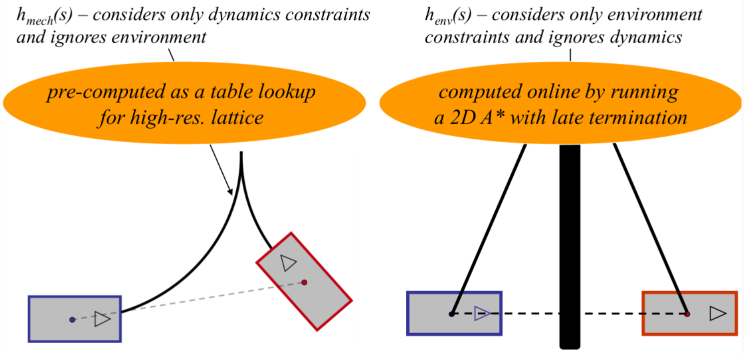


36 actions,	24 actions,
32 discrete values of heading	16 discrete values of heading
0.25m discretization for x,y	0.25m discretization for x,y

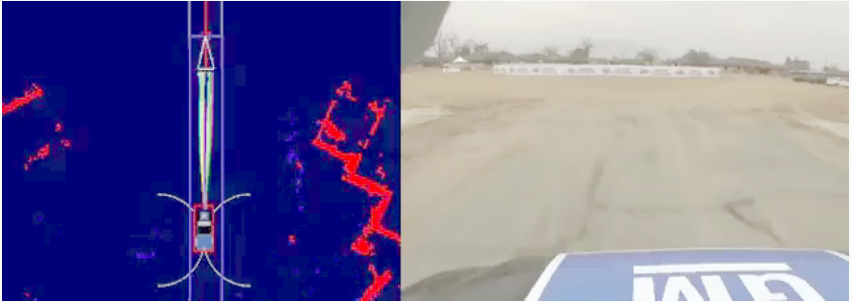
- Anytime D* to search (anytime and incrementally)

DARPA Urban Challenge (2007) [8]

- Heuristic $h(v) = \max(h_{mech}(v), h_{env}(v))$



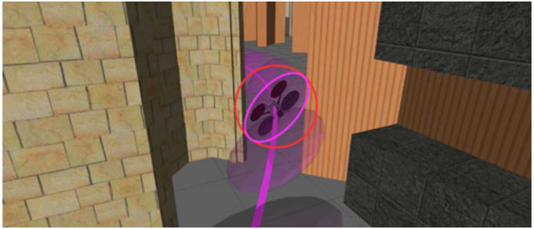
Video



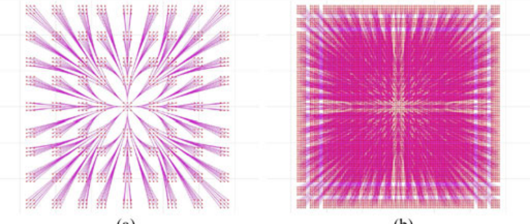
https://youtu.be/4hFhl0Oi8KI

Aggressive Quadrotor Flight (2018) [9]

- Goal: plan motions to fly through narrow gap

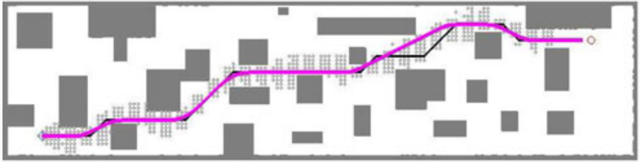


- Motion primitive generation: Quadrotors allow solving the two-value boundary problem efficiently (Domain knowledge)



Aggressive Quadrotor Flight (2018) [9]

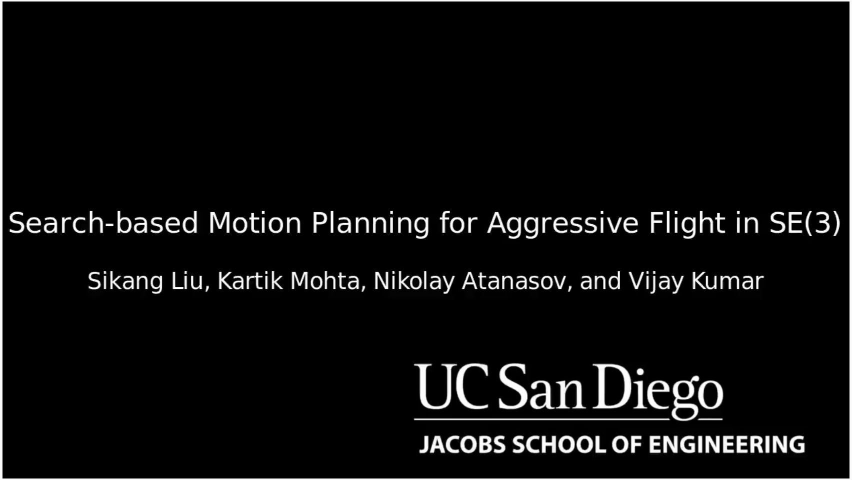
- Heuristic and Dimensionality: Use a hierarchical approach:
  1. Compute a solution in a low-dimensional state space (e.g., first-order model)
  2. Add a dimension to the state space (e.g., second-order model)
  3. Use first solution as heuristic



(a)

(b)

Video



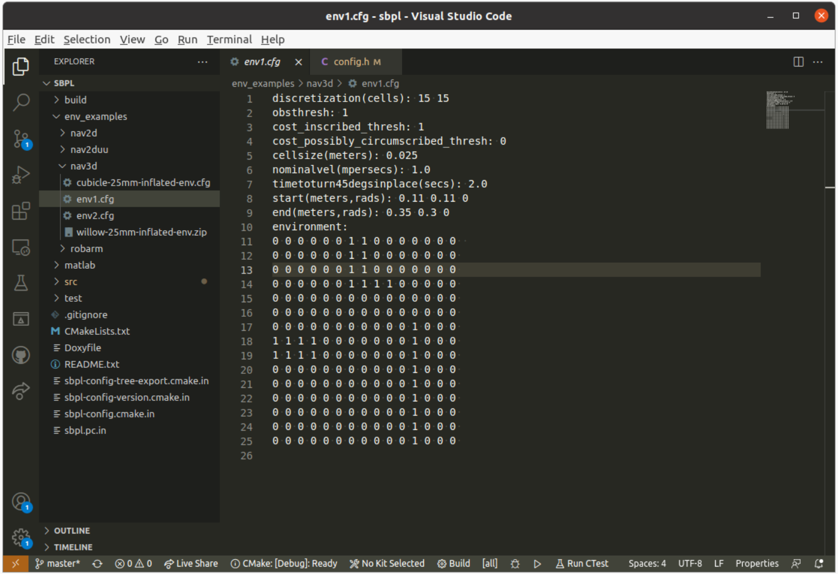
https://youtu.be/V4Mha-KPtwc

SBPL: Search-based Planning Library

Overview

- Open-Source, cross-platform C++ library: `https://github.com/sbpl/sbpl`
- Two parts:
    1. Graph Search (ARA*, Anytime D*)
    2. Environments (x, y, θ-lattice; manipulator)
- Some integration in the Robot Operating System (ROS)
- Heuristic/cost function: shortest path
- Motion primitives: examples for unicycle (MATLAB scripts)

Demo


env1.cfg - sbpl - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
SBPL
build
env_examples
nav2d
nav2duu
nav3d
cubicle-25mm-inflated-env.cfg
env1.cfg
env2.cfg
willow-25mm-inflated-env.zip
robarm
matlab
src
test
.gitignore
CMakeLists.txt
Doxyfile
README.txt
sbpl-config-tree-export.cmake.in
sbpl-config-version.cmake.in
sbpl-config.cmake.in
sbpl.pc.in
OUTLINE
TIMELINE
env1.cfg
config.h M
env_examples > nav3d > env1.cfg
discretization(cells): 15 15
obsthresh: 1
cost_inscribed_thresh: 1
cost_possibly_circumscribed_thresh: 0
cellsize(meters): 0.025
nominalvel(mpersecs): 1.0
timetoturn45degsinplace(secs): 2.0
start(meters,rads): 0.11 0.11 0
end(meters,rads): 0.35 0.3 0
environment:
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
1 1 1 1 0 0 0 0 0 0 0 1 0 0 0
1 1 1 1 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
master*
0 0
Live Share
CMake: [Debug]: Ready
No Kit Selected
Build
[all]
Run CTest
Spaces: 4
UTF-8
LF
Properties

Pros and Cons

+ Very fast
+ Implementation of advanced search-based planners
+ Used in ROS and MoveIt

- Very difficult to define new environments
- Very difficult to use custom set of motion primitives
- Not very active

Conclusion

A* Variants

- Anytime planning for realtime operation (wA*, focal search)
- Incremental for dynamic environments (ARA*)

State Lattices

- Enable kinodynamic planning
- Can re-use existing search-based planners
- Strong theoretical guarantees (up to discretization)
- Difficult to use for high-dimensional systems

Next Time

- Beginning of Part 3: Sampling-based Planning

Suggested Reading

1. Maxim Likhachev. *Planning and Decision-making in Robotics*. 2021. URL: http://www.cs.cmu.edu/~maxim/classes/robotplanning_grad/, Slide decks 5 and 6
2. Maxim Likhachev. *SBPL Tutorials*. URL: http://sbpl.net/Tutorials

References i

[1] Christopher Makoto Wilt and Wheeler Ruml. "When Does Weighted A* Fail?" In: *Symposium on Combinatorial Search, SOCS*. 2012. URL: http://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/view/5413.

[2] Ira Pohl. "Heuristic Search Viewed as Path Finding in a Graph". In: *Artif. Intell.* 1.3 (1970), pp. 193–204. DOI: 10.1016/0004-3702(70)90007-X.

[3] Judea Pearl and Jin H. Kim. "Studies in Semi-Admissible Heuristics". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 4.4 (1982), pp. 392–399. DOI: 10.1109/TPAMI.1982.4767270.

[4] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. “ARA*: Anytime A* with Provable Bounds on Sub-Optimality”. In: *Neural Information Processing Systems Conference (NIPS)*. MIT Press, 2004, pp. 767–774. URL: http://papers.nips.cc/paper/2382-ara-anytime-a-with-provable-bounds-on-sub-optimality.pdf.

[5] S. Koenig and M. Likhachev. “Fast Replanning for Navigation in Unknown Terrain”. In: *IEEE Transactions on Robotics* 21.3 (June 2005), pp. 354–363. ISSN: 1941-0468. DOI: 10.1109/TRO.2004.838026.

[6] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. “Anytime search in dynamic graphs”. In: *Artif. Intell.* 172.14 (2008), pp. 1613–1643. DOI: 10.1016/j.artint.2007.11.009.

[7] M. Pivtoraiko and A. Kelly. "Generating near Minimal Spanning Control Sets for Constrained Motion Planning in Discrete State Spaces". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2005, pp. 3231–3237. DOI: 10.1109/IROS.2005.1545046.

[8] Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. "Motion planning in urban environments". In: *J. Field Robotics* 25.11-12 (2008), pp. 939–960. DOI: 10.1002/rob.20265.

[9] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. "Search-Based Motion Planning for Aggressive Flight in SE(3)". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2439–2446. ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2795654.

[10] Maxim Likhachev. *Planning and Decision-making in Robotics*. 2021. URL: http://www.cs.cmu.edu/~maxim/classes/robotplanning_grad/.

[11] Maxim Likhachev. *SBPL Tutorials*. URL: http://sbpl.net/Tutorials.