New Perspectives on the Piano Movers’ Problem

Siddhartha Srinivasa
Boeing Endowed Professor
Personal Robotics Lab
University of Washington
Manipulation
Personal Robotics Lab
Carnegie Mellon University
Motion Planning is a technology

The Open Motion Planning Library

OMPL, the Open Motion Planning Library, consists of many state-of-the-art sampling-based motion planning algorithms. OMPL itself does not contain any code related to, e.g., collision checking or visualization. This is a deliberate design choice, so that OMPL is not tied to a particular collision checker or visualization front end. The library is designed so it can be easily integrated into systems that provide the additional needed components.

OMPLapp, the front-end for OMPL, contains a lightweight wrapper for the FCL and PQP collision checkers and a simple GUI based on PyQt/PySide. The graphical front-end can be used for planning motions for rigid bodies and a few vehicle types (first-order and second-order cars, a blimp, and a quadrotor). It relies on the Assimp library to import a large variety of mesh formats that can be used to represent the robot and its environment.
Formalizing the Core
Formalizing the Core

HRI

Machine Learning

Control
The Piano Movers’ Problem
Roadmaps
A* Search is Optimal …

Exploits Optimal Substructure
Best-first Search over Vertices
A* Search is **Optimal** …

Expands the Fewest Number of Vertices

But is this what we *really* want in Motion Planning?
Edge Evaluation Dominates Planning Time

Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. ICRA 2015
Edge Evaluation vs. Search
Explicit vs. Implicit Graphs
Is there a Search Algorithm that **Minimizes** the Number of Edge Evaluations?

**LazySP**

ICAPS 2018 [Best Conference Paper Award Winner]

First Provably Edge-Optimal A*-like Search Algorithm
LazySP
Greedy Best-first Search over Paths

To find the shortest path, eliminate all shorter paths!
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

F

Collision

Evaluate Path

Free

F

F
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

$F$
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

$F$
LazySP
Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path
Update the graph
Collision
Evaluate Path

Free

F

F
LazySP
Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path
Update the graph
Collision
Evaluate Path

F
Free

F
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Evaluate Path

Collision

Free

$F$

$F$
LazySP

Optimism Under Uncertainty

Graph, start, goal, lazy estimates

Lazy search for shortest path

Update the graph

Collision

Evaluate Path

Free

$F$
LazySP
Optimism Under Uncertainty
LazySP
Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty
LazySP

Optimism Under Uncertainty
Edge Selectors

**Forward**
(first unevaluated edge)

**Reverse**
(last unevaluated edge)

**Alternate**
(alternate Forward and Reverse)

**Bisect**
(furthest from an unevaluated edge)

Notes:
- Expand (all out-edges of frontier vertex)

Illustration:
- LazySP Selector

Path Distribution:
(most likely on a shortest path drawn from $D_p$)
The **Oracle** is a LazySP Selector!

The Oracle, LazySP, and A* algorithms are connected with edge evaluations. The Oracle provides the initial graph, start, and goal, along with lazy estimates. It then performs a lazy search for the shortest path. If there is a collision, it updates the graph; otherwise, it evaluates the path. Once the path is evaluated, it is added to the graph. The process continues until the goal is reached or all paths are evaluated. The edge evaluations are performed as the Oracle, LazySP, and A* algorithms interact.
The Story Continues ...

Hauser, Kris., Lazy collision checking in asymptotically-optimal motion planning. ICRA 2015
The Story Continues …

Graph Operations

Evaluations
Is there a Search Algorithm that **Minimizes** the Number of Edge Evaluations?

**LazySP**

ICAPS 2018 [Best Conference Paper Award Winner]

First Provably Edge-Optimal A*-like Search Algorithm
The Piano Movers' Problem
The **Experienced** Piano Movers’ Problem

New Piano.
New House.
Same Mover.
A Bayesian Approach to Edge Evaluation

Ground truth

Agent’s belief
Approach: Offline decision tree via DiRECT

Dataset

Problem is NP-Hard (Javdani et al. '14)

DiRECT (Chen et al. '15) frames it as a graph cut problem
Greedy policy ($O(|E| \Xi |\Phi|)$) is near-optimal

Executing DiRECT online is expensive, cached as a decision tree
Problem: What if test world not in dataset?

This can happen because we cannot enumerate all $\mathcal{O}(2^{|E|})$ worlds.

Solution: We developed BiSECT [NIPS’17] that can reason about all worlds implicitly.

1. BiSECT assumes edges are independent Bernoulli r.v.

2. BiSECT has linear complexity: $\mathcal{O}(|E||\Xi|)$

3. We execute BiSECT from the leaf of the tree
DIRECT + BISECT exploits structure
The Piano Movers’ Problem
Formalizing the Core

HRI

Machine Learning

Control
New Perspectives on the Piano Movers’ Problem

Siddhartha Srinivasa
Boeing Endowed Professor
Personal Robotics Lab
University of Washington