Using Machine Learning to improve Sampling in SBMPs

Tamir Offen
Student ID: 211621479
tamiroffen@campus.technion.ac.il

Hazem Irshed
Student ID: 211928569
Ir@campus.technion.ac.il

April 2024

1 Project Proposal - Draft

This project idea is based off of "Machine Learning Guided Exploration for Sampling-based Motion Planning Algorithms".

Main goal: learn to sample points from $\mathcal{X}_{rel}$, as the algorithm runs, not offline.
We want to do this because points in $\mathcal{X}_{rel}$ have the potential to be along an optimal path, when the heuristic is admissible.
Let $g : V \to \mathbb{R}_{>=0}$ be the cost to come function. $g(v)$ = path cost from x_{init} to v. $g^*(v)$ is the optimal cost to come function.

$$x_{goal} = argmin_{x \in \mathcal{X}_{goal}} g^*(x)$$

$$\mathcal{X}_{rel} = \{\underbrace{x \in \mathcal{X}_{free}}_{(1)} : \underbrace{g^*(x) + h(x) < g^*(x^*_{\text{goal}})}_{(2)}\}$$

How can we learn to sample points from $\mathcal{X}_{rel}$?
Break up sampling into two parts:

1. classification: $x_{sample} \overset{?}{\in} \mathcal{X}_{free}$, which is condition (1).
2. regression: predict $g^*(x_{sample})$, which is condition (2).

We can check if x_{sample} is in $\mathcal{X}_{rel}$ by checking both (1) and (2).

For the scope of this project, we will focus on learning condition (1), and learn condition (2) if we have time.

1.1 Learning the C-Space: $\mathcal{X}_{free}, \mathcal{X}_{obs}$

Problem statement: given our training set at iteration m, we want to find a function $\hat{y}_{\text{CS}} : \mathcal{X} \to \{\underbrace{-1}_{\text{col.}}, \underbrace{1}_{\text{col. free}}\}$.
The training set is $\mathcal{D} = \{\underbrace{(x^{(i)}, y^{(i)})}_{\text{config. and label at iter. i}} : i \in [m]\}$. Notice that our dataset increases with every iteration, this will make our model improve with time. Real collision checking is performed only for points that are classified as collision-free. We do this to not have to call the collision checker every time we sample a point, but we also do not want to have a situation where we include a point that is illegal (in collision) in our tree.

We want to use non-parametric classification, such as KDE (kernel density estimation). $\hat{y}_{\text{CS}}$ will be a Bayesian classifier that uses the estimated PDFs computed by KDE over the obstacle-free and obstacle points in the dataset. We decided not to use a model with weights because it would have to be trained over the entire dataset (and with a lot of upfront data), but we are building the dataset gradually as the algorithm runs. Link to video explaining KDE. The math can be found in the article section III D.

We learned that the edge validity checker is the most expensive part in SBMPs, so we can use $\hat{y}_{\text{CS}}$ also in the local planner for not calling the collision checker multiple times for each configuration. Question to Oren: does this make sense to do?(3)

We will use the environment in HW3, but with a robot with 2 links (so that the c-space will be 2D). We will set up an environment with a c-space that is filled with narrow passages. Show the distribution of points at different number of iterations and compare it with a random uniform distribution.

1.2 Higher Dimension Environment

If the answer to (3) is yes, then we will match our algorithm to the UR5e robot and it's environment and check if it will improve running time.

1.3 Learning the Cost-to-come Function $g^* : V \to \mathbb{R}_{>=0}$

Problem statement: given our training set at iteration m, we want to find a function $\hat{y}_{\text{CV}} : \mathcal{X} \to \mathcal{R}_{>=0}$ that will give a good estimate as the cost-to-go function.

Will implement this if time permits (most likely not because this section is quite complicated). There are implementation details in III.E.