Appendix for learning value functions with relational state representations for guiding task-and-motion planning

Beomjoon Kim **Luke Shimanuki**
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
{beomjoon,lukeshim}@mit.edu

1 Learning curve comparing Bellman-error and Large margin loss

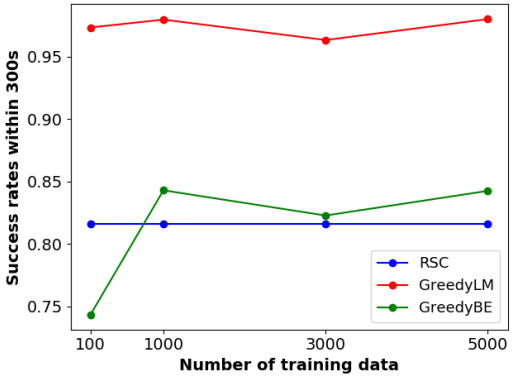


Figure 1: Learning curve of success rates for packing 1 box given 300s time limit in the box-moving domain

2 Details of GNN architecture

For all $f(\cdot, \theta_i)$, we use two fully-connected layers, each of which has 32 nodes. We use ReLU activation function for all nodes, except at the nodes of the last layer in $f(\cdot, \theta_4)$ and thatof $f(\cdot; \theta_5)$, which have the linear activation function. All layers have bias terms. We used Keras and tensorflow to implement the GNN. We perform two rounds of message passing in all of our experiments.

3 Predicate Evaluation and Caching

3.1 Predicate Implementations

The set of predicates we propose can apply to a wide range of geometric problem domains and operator classes. However, the manner of computing each predicate will vary between operator classes. We briefly describe the implementation details for each predicate in each domain we evaluated. We use the OpenRave inverse kinematics (IK) solver for the PR2 arms (given a sampled (x, y, θ) base pose for the robot). We also use a hand-built Probabilistic RoadMap (PRM) for motion planning between specified (x, y, θ) base poses, with fixed arm configurations.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

3.1.1 Box-Moving Domain: Two-Arm Pick-and-Place

- PREFREE(b): Sample feasible robot base poses and inverse kinematic solutions for picking b, then call the motion planner to find a collision-free path to any sampled base pose. This predicate is true if a collision-free path can be found.
- MANIPFREE(b, r): Sample feasible placements for b in region r, as well as robot base poses and inverse kinematic solutions, then call the motion planner to find a collision-free path from any pick base pose to any sampled place base pose. This predicate is true if a collision-free path can be found.
- OCCLUDESPRE(b_1, b_2): Sample feasible robot base poses and inverse kinematic solutions for picking b_2, then call the motion planner to find a collision-free path to any sampled base pose. If no such path can be found, instead find a path that ignores collisions. Then this predicate is true if the selected path collides with b_1.
- OCCLUDESMANIP(b_1, b_2, r): Sample feasible placements for b_2 in region r, as well as robot base poses and inverse kinematic solutions, then call the motion planner to find a collision-free path from any pick base pose to any sampled place base pose. If no such path can be found, instead find a path that ignores collisions. Then this predicate is true if the selected path collides with b_1.

3.1.2 Cupboard Domain: One-Arm Pick-and-Place

- PREFREE(b): Sample feasiblerobot base poses and inverse kinematic solutions for picking b. This predicate is true if a collision-free arm solution can be found.
- MANIPFREE(b, r): Sample feasible placements for b in region r, as well as robot base poses and inverse kinematic solutions. This predicate is true if a collision-free solution can be found.
- OCCLUDESPRE(b_1, b_2): Sample feasible robot base poses and inverse kinematic solutions for picking b_2. If no collision-free solution can be found, instead find a solution that ignores collisions. Then this predicate is true if the selected arm configuration collides with b_1.
- OCCLUDESMANIP(b_1, b_2, r): Sample feasible placements for b_2 in region r, as well as robot base poses and inverse kinematic solutions. If no collision-free solution can be found, instead find a solution that ignores collisions. Then this predicate is true if the selected arm configuration collides with b_1.

3.2 Caching

Evaluating each predicate is usually a very expensive operation, but there is a lot of information that can be retained across different calls, or across different iterations within the same call, to the predicate evaluation function. Therefore, we use caching extensively to make the repeated computation of the predicates efficient. Although these techniques are tied to our specific implementation, the approach is quite general.

3.2.1 Probabilistic Roadmap for Motion Planning

A motion planner is called to sample the continuous operator parameters in the box-moving domain. It is also used to evaluate the predicates PREFREE, MANIPFREE, OCCLUDESPRE, and OCCLUDESMANIP. In our environment, the walls and other fixed objects remain constant across all problem instances, as only the movable objects have varying initial poses. Therefore, we pre-compute a finely-sampled probabilistic roadmap (PRM) that ignores movable objects but respects fixed objects. Later, when doing the graph search for a path, we check for collisions for motions on the edges of the PRM against the movable objects in the state. This leads to more efficient and less variable motion planning calls.

3.2.2 Cached Collisions and Paths

In a single state we make many motion planning calls. Performing collision checks between movable objects and robot configurations for each motion planning call can be quite expensive. So we cache which configurations in the PRM collide with each object in the current state, then reuse that information in future graph searches. We also retain collision-free paths that are reused in multiple predicate evaluations.

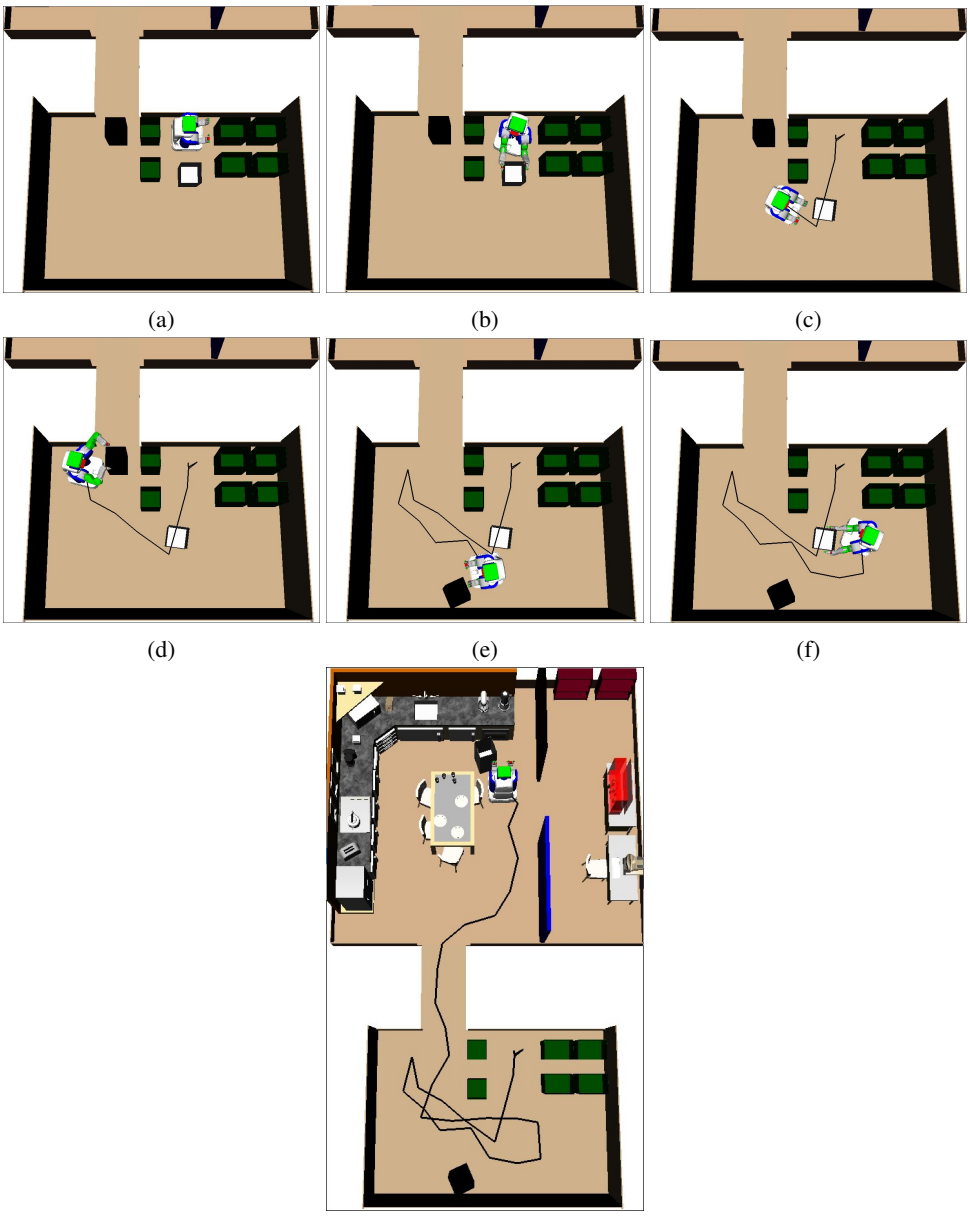
3.2.3 Inverse Kinematics Solutions

Unlike in the box-moving domain, in which collisions mostly constrain the space of feasible trajectories of the robot base, collisions in the cupboard domain heavily constrain the space of feasible arm configurations instead. Therefore, many inverse kinematic solutions are in collision, and so we must sample many configurations in order to find a feasible operator. Because inverse kinematic solving is a relatively expensive operation, this severely impacts the efficiency of planning and evaluating predicates. The workaround we use is to pre-compute a large number of inverse kinematic solutions for objects at a wide variety of poses relative to the robot base. Then at planning time we adapt the cached configuration by using relative transformations to make it fit with the actual object pose (for a pick operation) or the desired object placement (for a place operation). Many of these cached solutions will still be in collision, but avoiding the cost of finding the kinematic solution leads to significant speedups.

3.2.4 Predicate Evaluations

Finally, when an action is applied to a state, resulting in a new state, a lot of information can be passed down to improve the efficiency of evaluating the predicates for the new state. First, for any given action, many predicates will not change because moving a single object will leave most relationships between other objects the same, so we reuse the predicate value without recomputing it. Additionally, the set of PRM configurations in collision with each object only changes for the object that was moved, and so all other sets of collisions can be reused. In both domains, inverse kinematics solutions that are known to not collide with fixed objects can be retained for all objects except for the one that was moved. These configurations might still collide with movable objects, though.

4 Solving Nonmonotonic Problem Instances



(a) (b) (c)

(d) (e) (f)

(g)

Figure 2: An example of a problem instance that is nonmonotonic, in that the robot cannot move the goal object (white) to the goal region (the upper region) by only touching each object at most once. In this particular instance, the black box is blocking the corridor to the goal region, so the robot cannot move the white box directly to the goal region. However, the white box is blocking the robot from reaching the black box. Therefore, the robot must move the white box out of the way in order for it to move the black box, then it must touch the white box again to move it to the goal. Nonmonotonic problem instances such as this cannot be solved by Stillman's algorithm by design, but the greedy search planner with our learned Q function finds a plan in 30.4 seconds, averaged across 6 training and 20 planning seeds. This is interesting because it is able to learn to solve nonmonotonic problems even though it was trained on planning experience only from monotonic problems.