Sub-Goal Trees – a Framework for Goal-Based Reinforcement Learning

Tom Jurgenson [1] **Or Avner** [1] **Edward Groshev** [2] **Aviv Tamar** [1]

Abstract

Many AI problems, in robotics and other domains, are goal-based, essentially seeking trajectories leading to various goal states. Reinforcement learning (RL), building on Bellman's optimality equation, naturally optimizes for a single goal, yet can be made multi-goal by augmenting the state with the goal. Instead, we propose a new RL framework, derived from a dynamic programming equation for the *all pairs shortest path* (APSP) problem, which naturally solves multi-goal queries. We show that this approach has computational benefits for both standard and approximate dynamic programming. Interestingly, our formulation prescribes a novel protocol for computing a trajectory: instead of predicting the next state given its predecessor, as in standard RL, a goal-conditioned trajectory is constructed by first predicting an intermediate state between start and goal, partitioning the trajectory into two. Then, recursively, predicting intermediate points on each sub-segment, until a complete trajectory is obtained. We call this trajectory structure a *sub-goal tree*. Building on it, we additionally extend the policy gradient methodology to recursively predict sub-goals, resulting in novel goal-based algorithms. Finally, we apply our method to neural motion planning, where we demonstrate significant improvements compared to standard RL on navigating a 7-DoF robot arm between obstacles.

1. Introduction

Many AI problems can be characterized as learning or optimizing goal-based trajectories of a dynamical system, for example, robot skill learning and motion planning (Mülling et al., 2013; Gu et al., 2017; LaValle, 2006; Qureshi et al., 2018). The reinforcement learning (RL) formulation, a popular framework for trajectory optimization based on Bellman's dynamic programming (DP) equation (Bertsekas & Tsitsiklis, 1996), naturally addresses the case of a single goal, as specified by a single reward function. RL formulations for multiple goals have been proposed (Schaul et al., 2015; Andrychowicz et al., 2017), typically by augmenting the state space to include the goal as part of the state, but without changing the underlying DP structure.

On the other hand, in deterministic shortest path problems, multi-goal trajectory optimization is most naturally represented by the all-pairs shortest path (APSP) problem (Russell & Norvig, 2010). In this formulation, augmenting the state and using Bellman's equation is known to be sub-optimal[1], and dedicated APSP algorithms such as Floyd-Warshall (Floyd, 1962) build on different DP principles. Motivated by this, we propose a goal-based RL framework that builds on an efficient APSP solution, and is also applicable to large or continuous state spaces using function approximation.

Our key idea is that a goal-based trajectory can be constructed in a divide-and-conquer fashion. First, predict a sub-goal between start and goal, partitioning the trajectory into two. Then, recursively, predict intermediate sub-goals on each sub-segment, until a complete trajectory is obtained. We call this trajectory structure a *sub-goal tree* (Figure 1), and we develop a DP equation for APSP that builds on it and is compatible with function approximation. We further bound the error of following the sub-goal tree trajectory in the presence of approximation errors, and show favorable results compared to the conventional RL method, intuitively, due to the sub-goal tree's lower sensitivity to drift.

The sub-goal tree can also be seen as a general parametric structure for a trajectory, where a parametric model, e.g., a neural network, is used to predict each sub-goal given its predecessors. Based on this view, we develop a policy gradient framework for sub-goal trees, and show that conventional policy-gradient techniques such as control variates and trust regions (Greensmith et al., 2004; Schulman et al., 2015) naturally apply here as well.

Finally, we present an application of our approach to neu-

[1]EE Department, Technion [2]Osaro Inc.. Correspondence to: Tom Jurgenson <tomj@campus.technion.ac.il>, Aviv Tamar <avivt@technion.ac.il>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

[1]This is equivalent to running the Bellman-Ford single-goal algorithm for each possible goal state (Russell & Norvig, 2010).

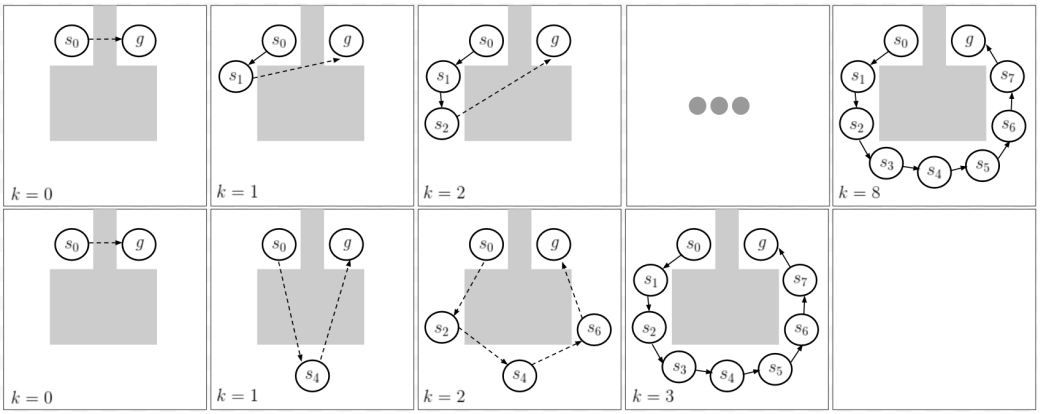


Figure 1. Trajectory prediction methods. Upper row: a conventional *Sequential* representation. Lower row: *Sub-Goal Tree* representation. Solid arrows indicate predicted segments, while dashed arrows indicate segments that still require to be predicted. By concurrently predicting sub-goals, a *Sub-Goal Tree* only requires 3 sequential computations, while the *sequential* requires 8.

ral motion planning – learning to navigate a robot between obstacles (Qureshi et al., 2018; Jurgenson & Tamar, 2019). Using our policy gradient approach, we demonstrate navigating a 7-DoF continuous robot arm safely between obstacles, obtaining marked improvement in performance compared to conventional RL approaches.

2. Related work

In RL, the idea of sub-goals has mainly been investigated under the options framework (Sutton et al., 1999). In this setting, the goal is typically fixed (i.e., given by the reward in the MDP), and useful options are discovered using some heuristic such as bottleneck states (McGovern & Barto, 2001; Menache et al., 2002) or changes in the value function (Konidaris et al., 2012). While hierarchical RL using options seems intuitive, theoretical results for their advantage are notoriously difficult to establish, and current results require non-trivial assumptions on the option structure (Mann & Mannor, 2014; Fruit et al., 2017). Interestingly, by investigating the APSP setting, we obtain general and strong results for the advantage of sub-goals.

Universal value functions (Schaul et al., 2015; Andrychowicz et al., 2017) learn a goal-conditioned value function using the Bellman equation. In contrast, in this work we propose a principled motivation for sub-goals based on the APSP problem, and develop a new RL formulation based on this principle. A connection between RL and the APSP has been suggested by Kaelbling (1993); Dhiman et al. (2018), based on the Floyd-Warshall algorithm. However, as discussed in Section 4.2, these approaches become unstable once function approximation is introduced. Goal-conditioned policies are also related to universal plans in the classical planning literature (Schoppers, 1989; Kaelbling, 1988; Ginsberg, 1989), and in this sense SGTs can be used to approximate universal plans using learning.

Sub-goal trees can be seen as a form of trajectory representation (as we discuss in Section 5.2). Such has been investigated for learning robotic skills (Mülling et al., 2013; Sung et al., 2018) and navigation (Qureshi et al., 2018). A popular approach, dynamical movement primitives (Ijspeert et al., 2013), represents a continuous trajectory as a dynamical system with an attractor at the goal, and was successfully used for RL (Kober et al., 2013; Mülling et al., 2013; Peters & Schaal, 2008). The temporal segment approach of Mishra et al. (2017), on the other hand, predicts segments of a trajectory sequentially. In the context of video prediction, Jayaraman et al. (2019) proposed to predict salient frames in a goal-conditioned setting by a supervised learning loss that focuses on the 'best' frames. This was used to predict a list of sub-goals for a tracking controller. In contrast, we investigate the RL problem, and predict sub-goals *recursively*.

Approximate APSP solutions have been traditionally explored in the non-learning setting. Approaches such as in Thorup & Zwick (2005); Chan (2010); Williams (2014) compute k-optimal paths ($k > 1$) in running time that is sub-cubic in the number of vertices in the graph. These methods require full knowledge of the graph and process each vertex at least once. In contrast, our learning-based approach can generalize to unseen states (vertices) by learning from similar states, potentially scaling to very large or even continuous state spaces, as we demonstrate in our experiments.

3. Problem Formulation and Notation

We are interested in optimizing goal-conditioned tasks in dynamical systems. We restrict ourselves to deterministic, stationary, and finite time systems, and consider both discrete and continuous state formulations.[2]

[2]The deterministic setting is fundamental to our approach, and therefore we do not build on the popular Markov decision pro-

In the discrete state setting, we study the all-pairs shortest path (APSP) problem on a graph. Consider a directed and weighted graph with N nodes $s_1, \ldots, s_N$, and denote by $c(s, s') \geq 0$ the weight of edge $s \to s'$.[3] By definition, we set $c(s, s) = 0$, such that the shortest path from a node to itself is zero. To simplify notation, we can replace unconnected edges by edges with weight ∞. Thus, without loss of generality, throughout this work we assume that the graph is complete. The APSP problem seeks the shortest paths (i.e., a path with minimum sum of costs) **from any start node** s **to any goal node** g in the graph:

$$\min_{T, s_0=s, s_1, \ldots, s_{T-1}, s_T=g} \sum_{t=0}^{T-1} c(s_t, s_{t+1}). \tag{1}$$

In the continuous state case, we consider a deterministic controlled dynamical system in discrete time, defined over a state space S and an action space U: where $s_t, s_{t+1} \in S$ are the states at time t and $t+1$, respectively, $u_t \in U$ is the control at time t, and f is a stationary state transition function. Given an initial state s_0 and a goal g, an optimal trajectory reaches g while minimizing the sum of non-negative costs $\bar{c}(s, u) \geq 0$:

$$\min_{T, u_0, \ldots, u_T} \sum_{t=0}^{T} \bar{c}(s_t, u_t), \text{ s.t. } s_{t+1} = f(s_t, u_t), s_T = g. \tag{2}$$

To unify our treatment of problems (1) and (2), in the remainder of this paper we abstract away the actions as follows. Let $c(s, s') = \min_u \bar{c}(s, u)$ s.t. $s' = f(s, u)$, and $c(s, s') = \infty$ if no transition from s to s' is possible. Then, for any start and goal states s, g, Eq. (2) is equivalent to:[4]

$$\min_{T, s_0=s, s_1, \ldots, s_{T-1}, s_T=g} \sum_{t=0}^{T-1} c(s_t, s_{t+1}). \tag{3}$$

Note the similarity of Problem (3) to the APSP problem (1), where feasible transitions of the dynamical system are represented by edges between states in the graph. In the rest of this paper, we study solutions for (1) and (3), for all start and goal pairs s, g. We are particularly interested in large problems, where exact solutions are intractable, and approximations are required.

Notation: Let $\tau = s_0, \ldots, s_T$ denote a state trajectory. Also, denote $c_{i:j} = \sum_{t=i}^{j-1} c(s_t, s_{t+1})$, the cost of several subsequent states and $c_\tau = c_{0:T}$ as the total trajectory cost. Let $\tau(s_0, s_m) = s_0, \ldots, s_m$ denote the segment of τ starting at s_0 and ending at s_m. We denote the trajectory concatenation as $\tau = [\tau(s_0, s_m), \tau(s_m, s_T)]$, where it is implied that s_m appears only once. For a stochastic trajectory distribution conditioned on start and goal, we denote $\Pr_\pi[s_i, \ldots, s_j | s, g]$ as shorthand for $\Pr_\pi[s_i, \ldots, s_j | s_i = s, s_j = g]$.

cess framework. We leave the investigation of similar ideas to stochastic and time varying systems for future work. We note that deterministic systems are popular in RL applications, such as Bellemare et al. (2013); Duan et al. (2016).

[3] Technically, and similarly to standard APSP algorithms (Russell & Norvig, 2010), we only require that there are no negative cycles in the graph. To simplify our presentation, however, we restrict c to be non-negative.

[4] In practice, and as we report in our experiments, such an abstraction can be easily implemented using an inverse model.

4. Approximate Dynamic Programming

In this section we study a dynamic programming principle for the APSP problem (1). We will later extend our solution to the continuous case (3) using function approximation.

One way to solve (1) is to solve a single-goal problem for every possible goal state. The RL equivalent of this approach is UVFA, suggested by Schaul et al. (2015), where the state space is augmented with the goal state, and a goal-based value function is introduced. The UVFA is optimized using standard (single-goal) RL, by learning over different goals. Alternatively, our approach builds on an efficient APSP dynamic programming algorithm, as we propose next.

4.1. Sub-Goal Tree Dynamic Programming

By our definition of non-negative costs, the shortest path between any two states is of length at most N. The main idea in our approach is that a trajectory between s and g can be constructed in a divide-and-conquer fashion: first predict a sub-goal between start and goal that optimally partitions the trajectory to two segments of length $N/2$ or less: $s, \ldots, s_{N/2}$ and $s_{N/2}, \ldots, g$. Then, recursively, predict intermediate sub-goals on each sub-segment, until a complete trajectory is obtained. We term this composition a *sub-goal tree* (SGT), and we formalize it as follows.

Let $V_k(s, s')$ denote the shortest path from s to s' in 2^k steps or less. Note that by our convention about unconnected edges above, if there is no such trajectory in the graph then $V_k(s, s') = \infty$. We observe that V_k obeys the following dynamic programming relation, which we term *sub-goal tree dynamic programming* (SGTDP).

Theorem 1. *Consider a weighted graph with N nodes and no negative cycles. Let $V_k(s, s')$ denote the cost of the shortest path from s to s' in 2^k steps or less, and let $V^*(s, s')$ denote the cost of the shortest path from s to s'. Then, V_k can be computed according to the following equations:*

$$\begin{aligned} &V_0(s, s') = c(s, s'), \quad \forall s, s' : s \neq s'; \\ &V_k(s, s) = 0, \quad \forall s; \\ &V_k(s, s') = \min_{s_m} \{V_{k-1}(s, s_m) + V_{k-1}(s_m, s')\}, \forall s, s' : s \neq s'. \end{aligned} \tag{4}$$

Furthermore, for $k \geq \log_2(N)$ we have that $V_k(s, s') = V^*(s, s')$ for all s, s'.

The proof of Theorem 1, along with all other proofs in this paper, is reported in the supplementary material.

Given $V_0, \ldots, V_{\log_2(N)}$, Theorem 1 prescribes the following recipe for computing a shortest path for every s, g, which we term the **greedy SGT trajectory**:[5]

$$
\begin{aligned}
&s_0 = s,\ s_N = g \qquad (5)\\
&s_{N/2} \in \arg\min_{s_m} \left\{V_{\log_2(N)-1}(s_0, s_m) + V_{\log_2(N)-1}(s_m, s_N)\right\},\\
&s_{N/4} \in \arg\min_{s_m} \left\{V_{\log_2(N)-2}(s_0, s_m) + V_{\log_2(N)-2}(s_m, s_{N/2})\right\},\\
&s_{3N/4} \in \arg\min_{s_m} \left\{V_{\log_2(N)-2}(s_{N/2}, s_m) + V_{\log_2(N)-2}(s_m, s_N)\right\},\\
&\ldots
\end{aligned}
$$

The SGT can be seen as a general divide-and-conquer representation of a trajectory, by recursively predicting sub-goals. In contrast, the construction of a greedy trajectory in conventional RL (using the Bellman equation) proceeds *sequentially*, by predicting state (or action) after state. This is illustrated in Figure 1. We next investigate the advantages of the SGT structure compared to the sequential approach.

In conventional RL, it is well known that in the presence of errors, following a greedy policy may suffer from *drift* – accumulating errors that may hinder performance (Ross et al., 2011). In the goal-based setting, intuitively, the error magnitude scales with the distance from the goal. Our main observation is that the SGT is less sensitive to drift, as the sub-goals break the trajectory into smaller segments with exponentially decreasing errors. We next formalize this idea, by analyzing an approximate DP formulation.

4.2. SGT Approximate DP

Similar to standard dynamic programming algorithms, the SGTDP algorithm iterates over all states in the graph, which becomes infeasible for large, or continuous state spaces. For such cases, inspired by the approximate dynamic programming (ADP) literature (Bertsekas, 2005), we investigate ADP methods based on SGTDP.

Let T denote the SGTDP operator, $(TV)(s, s') = \min_{s_m} \{V(s, s_m) + V(s_m, s')\}$. From Theorem 1, we have that $V^* = T^{\log_2(N)} V_0$. Similarly to standard ADP analysis (Bertsekas, 2005), we assume an ϵ-approximate SGTDP operator that generates a sequence of approximate value functions $\hat{V}_0, \ldots, \hat{V}_{\log_2(N)}$ that satisfy:

$$\|\hat{V}_{k+1} - T\hat{V}_k\|_\infty \leq \epsilon, \qquad \|\hat{V}_0 - V_0\|_\infty \leq \epsilon, \qquad (6)$$

where $\|x\|_\infty = \max_{s,s'} |x(s, s')|$. The next result provides an error propagation bound for SGTDP.[6]

Proposition 1. *For the sequence $\hat{V}_0, \ldots, \hat{V}_{\log_2(N)}$ satisfying (6), we have that $\|\hat{V}_{\log_2(N)} - V^*\|_\infty \leq \epsilon(2N - 1)$.*

A similar $O(\epsilon N)$ error bound is known for ADP based on the approximate Bellman operator (Bertsekas & Tsitsiklis, 1996, Pg. 332). Thus, Proposition 1 shows that given the same value function approximation method, we can expect a similar error in the value function between the SGT and sequential approaches. However, the main importance of the value function is in deriving a policy, in this case, a trajectory from start to goal. As we show next, the shortest path derived from the approximate SGTDP value function can be significantly more accurate than a path derived using the Bellman value function.

We first discuss how to compute a greedy trajectory. Given a sequence of ϵ-approximate SGTDP value functions $\hat{V}_0, \ldots, \hat{V}_{\log_2(N)}$ as described above, one can compute the greedy SGT trajectory by plugging the approximate value functions in (5) instead of their respective accurate value functions. We term this the **approximate greedy SGT trajectory**, and provide an error bound for it.

Proposition 2. *For a start and goal pair s, g, and sequence of value functions $\hat{V}_0, \ldots, \hat{V}_{\log_2(N)}$ generated by an ϵ-approximate SGTDP operator, let $s_0, \ldots, s_N$ denote the approximate greedy SGT trajectory as described above. We have that $\sum_{i=0}^{N-1} c(s_i, s_{i+1}) \leq V^*(s, g) + 4N \log_2(N)\epsilon$.*

Thus, the error of the greedy SGT trajectory is $O(N \log_2(N))$. In contrast, for the greedy trajectory according to the standard finite-horizon Bellman equation, a tight $O(N^2)$ bound holds (Ross et al., 2011, we also provide an independent proof in the supplementary material). Thus, the SGT approach provides a strong improvement in handling errors compared to the sequential method. In addition, the SGT approach requires us to compute and store only $\log_2(N)$ different value functions. In comparison, a standard finite horizon approach would require storing N value functions, which can be limiting for large N. Finally, during trajectory prediction, sub-goal predictions for different branches of the tree are independent and can be computed concurrently, allowing an $O(\log_2(N))$ prediction time, whereas sequentially predicting sub-goals is $O(N)$. We thus conclude that the SGT provides significant benefits in both approximation accuracy, prediction time, and space.

Why not use the Floyd-Warshall Algorithm? At this point, the reader may question why we do not build on the Floyd-Warshall (FW) algorithm for the APSP problem. The FW method maintains a value function $V_{FW}(s, s')$ of the shortest path from s to s', and

[5] Note that the cost of the greedy SGT trajectory is optimal, however, its length is by definition N. Thus, a shorter-length trajectory with the same cost may exist. Simple modifications of SGT to find the shortest-length cost optimal trajectory are possible, but we do not consider them here.

[6] For conventional RL, $\|\cdot\|_p$ errors bounds were developed by Munos & Szepesvári (2008), and are more suitable for the learning setting. We leave such investigation to future work, and emphasize that our simpler $\|\cdot\|_\infty$ bounds still show the fundamental soundness of our approach.

updates the value using the relaxation $V_{FW}(s,s') := \min\{V_{FW}(s,s'), V_{FW}(s,s_m)+V_{FW}(s_m,s')\}$. If the updates are performed over all s_m, s, and s' (in that sequence), V_{FW} will converge to the shortest path, requiring $O(N^3)$ computations, compared to $O(N^3 \log N)$ for SGTDP (Russell & Norvig, 2010). One can also perform relaxations in an arbitrary order, as was suggested by Kaelbling (1993), and more recently by Dhiman et al. (2018), to result in an RL style algorithm. However, as was already observed by Kaelbling (1993), the FW relaxation requires that the values always over-estimate the optimal costs, and any under-estimation error, due to noise or function approximation, gets propagated through the algorithm without any way of recovery, leading to instability. Our SGT approach avoids this problem by updating V_k based on V_{k-1}, resembling finite horizon dynamic programming, and, as we proved, maintains stability in presence of errors. Furthermore, both Kaelbling (1993); Dhiman et al. (2018) showed results only for table-lookup value functions. In our experiments, we have found that replacing the SGTDP update with a FW relaxation (described in the supplementary) leads to instability when used with function approximation.

5. Sub-Goal Tree RL Algorithms

Motivated by the theoretical results of the previous section, we present algorithms for learning SGT policies. We start with a value-based batch-RL algorithm with function approximation in Section 5.1. Then, in Section 5.2, we present a policy gradient approach for SGTs.

5.1. Batch RL with Sub-Goal Trees

We now describe a batch RL algorithm with function approximation based on the SGTDP algorithm above. Our approach is inspired by the fitted-Q iteration (FQI) algorithm for finite horizon Markov decision processes (Tsitsiklis & Van Roy 2001; see also Ernst et al. 2005; Riedmiller 2005 for the discounted case). Similar to FQI, we are given a data set of M random state transitions and their costs $\{(s_i, s'_i, c_i)\}_{i=1}^M$, and we want to estimate $V_k(s,s')$ for arbitrary pairs (s,s'). Assume that we have some estimate $\hat{V}_k(s,s')$ of the value function of depth k in SGTDP. Then, for any pair of start and goal states s, g, we can estimate $\hat{V}_{k+1}(s,g)$ as

$$\hat{V}_{k+1}(s,g) = \min_{s_m} \left\{ \hat{V}_k(s,s_m) + \hat{V}_k(s_m,g) \right\}. \quad (7)$$

Thus, if our data consisted of start and goal pairs, we could use (7) to generate *regression targets* for the next value function, and use any regression algorithm to fit $\hat{V}_{k+1}(s,s')$. This is the essence of the Fitted SGTDP algorithm (Algorithm 1). Since our data does not contain explicit goal states, we simply define goal states to be randomly selected states from within the data (lines 6,7 in Alg. 1).

The first iteration $k=0$ in Fitted SGTDP, however, requires special attention. We need to fit the cost function for connected states in the graph, yet make sure that states which are not reachable in a single transition have a high cost. To this end, we fit the observed costs c to the observed transitions s, s' in the data (lines 2,5 in Alg. 1), and a high cost C_{max} to transitions from the observed states to randomly selected states (lines 3,5 in Alg. 1). We also fit a cost of zero to self transitions (lines 4,5 in Alg. 1).

Our algorithm also requires a method to approximately solve the minimization problem in (7). In our experiments, we discretized the state space and performed a simple grid search. Other methods could be used in general. For example, if V_k is represented as a neural network, then one can use gradient descent. Naturally, the quality of Fitted SGTDP will depend on the quality of solving this minimization problem.

Algorithm 1 Fitted SGTDP

Algorithm

1 Input: dataset $D = \{s, c, s'\}$, max path cost C_{max}
2 Create real transition data:
$D_{trans} = \{s, s'\}$ with targets $T_{trans} = \{c\}$ from D
3 Create fake transition data:
$D_{random} = \{s, s_{rand}\}$ with targets $T_{random} = \{C_{max}\}$ with s, s_{rand} random states from D
4 Create self transition data:
$D_{self} = \{s, s\}$ with targets $T_{self} = \{0\}$ with s taken from D
5 Fit $\hat{V}_0(s,s')$ to data in $D_{trans}, D_{random}, D_{self}$ and targets $T_{trans}, T_{random}, T_{self}$
for $k : 1...K$ **do**
6 Create goal data $D_{goal} = \{s, g\}$ and targets $T_{goal} = \{\min_{s_m}\{\hat{V}_{k-1}(s,s_m) + \hat{V}_{k-1}(s_m,g)\}\}$ with s, g randomly chosen from states in D
7 Fit $\hat{V}_k(s,s')$ to data in D_{goal} and targets in T_{goal}
end

5.2. Sub-Goal Trees Policy-Gradient

The minimization over sub-goal states in Fitted SGTDP can be difficult for high-dimensional state spaces. A similar problem arises in standard RL with continuous actions (Bertsekas, 2005; Kalashnikov et al., 2018), and has motivated the study of policy search methods, which directly optimize the policy (Deisenroth et al., 2013). Following a similar motivation, we propose a policy search approach based on SGTs. Inspired by policy gradient (PG) algorithms in conventional RL (Sutton et al., 2000; Deisenroth et al., 2013), we propose a parametrized stochastic policy that approximates the optimal sub-goal prediction, and we develop a corresponding PG theorem for training the policy.

Stochastic SGT policies: A stochastic SGT policy

$\pi(s'|s_1, s_2)$ is a stochastic mapping from two endpoint states $s_1, s_2 \in S$ to a predicted sub-goal $s' \in S$. Given a start state $s_0 = s$ and goal $s_T = g$, the likelihood of $\tau = s_0, s_1, \ldots, s_T$ under policy π is defined recursively by:

$$\begin{aligned}&\Pr_{\pi}[\tau|s,g] = \Pr_{\pi}[s_0,\ldots,s_T|s,g] \\ &= \Pr_{\pi}[s_0,\ldots,s_{\frac{T}{2}}|s,s_m]\Pr_{\pi}[s_{\frac{T}{2}},\ldots,s_T|s_m,g]\pi(s_m|s,g),\end{aligned} \tag{8}$$

where the base of the recursion is $\Pr_{\pi}[s_t, s_{t+1}|s_t, s_{t+1}] = 1$ for all $t \in [0, T-1]$. This formulation assumes that sub-goal predictions within a segment depend only on states within the segment, and not on states before or after it. This is analogous to the Markov property in conventional RL, where the next state prediction depends only on the previous state, but adapted to a goal-conditioned setting.

Note that this recursive decomposition can be interpreted as a tree. Without loss of generality, we assume this tree has a depth of D thus $T = 2^D$ (repeating states to make the tree a full binary tree does not incur extra cost as $c(s, s) = 0$).

We now define the PG objective. Let ρ_0 denote a distribution over start and goal pairs $s, g \in S$. our goal is to learn a stochastic policy $\pi_\theta(s'|s_1, s_2)$, characterized by a parameter vector θ, that minimizes the expected trajectory costs:

$$J(\theta) = J^{\pi_\theta} = \mathbb{E}_{\tau\sim\rho(\pi_\theta)}\left[c_\tau\right], \tag{9}$$

where the trajectory distribution $\rho(\pi_\theta)$ is defined by first drawing $(s, g) \sim \rho_0$, and then recursively drawing intermediate states as defined by Eq. (8). Our next result is a PG theorem for SGTs.

Theorem 2. *Let π_θ be a stochastic SGT policy, $\rho(\pi_\theta)$ be a trajectory distribution defined above, and $T = 2^D$. Then*

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\rho(\pi_\theta)}\left[c_\tau \cdot \nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau]\right] \\ &= \mathbb{E}_{\rho(\pi_\theta)}\left[\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}} C_\tau^{i,d}\cdot\nabla_\theta \log \pi_\theta\left(s_m^{i,d}\middle|s^{i,d}, g^{i,d}\right)\right],\end{aligned} \tag{10}$$

where $s^{i,d} = s_{(i-1)\cdot 2^d}$, $s_m^{i,d} = s_{(2i-1)\cdot 2^{d-1}}$, $g^{i,d} = s_{i\cdot 2^d}$, and $C_\tau^{i,d} = c_{(i-1)\cdot 2^d : i\cdot 2^d}$ is the sum of costs from $s^{i,d}$ to $g^{i,d}$ of τ. Furthermore, let the baseline $b^{i,d} = b(s^{i,d}, g^{i,d})$ be any fixed function $b^{i,d} : S^2 \to R$, then $C_\tau^{i,d}$ in (10) can be replaced with $C_\tau^{i,d} - b^{i,d}$.

The main difference between Theorem 2 and the standard PG theorem (Sutton et al., 2000) is in the calculation of $\nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau]$, which in our case builds on the tree-based construction of the SGT trajectory. The summations over i and d in Theorem 2, are used to explicitly state individual sub-goal predictions: d iterates over the depth of the tree, and i iterates between sub-goals of the same depth.

Using Theorem 2, we can sample a trajectory τ using π_θ, obtain its cost c_τ, and estimate of the gradient of $J(\theta)$ using c_τ and the gradients of individual decisions of π_θ.

In the policy gradients literature, variance reduction using control variates, and trust-region methods such as TRPO and PPO play a critical role (Greensmith et al., 2004; Schulman et al., 2015; 2017). The baseline reduction in Theorem 2 allows similar results to be derived for SGT, and we similarly find them to be important in practice. Due to space constraints, we report these in the supplementary.

5.3. The SGT-PG Algorithm

Algorithm 2 SGT-PG

Algorithm

Input: D - depth, N - episodes per cycle, E - environment
init $\pi_1, \ldots, \pi_D$ with parameters $\theta_1, \ldots, \theta_D$
for $d : 1...D$ **do**
  if $d > 1$ **then**
    $\theta_d \leftarrow \theta_{d-1}$ `// init` π_d `from` π_{d-1}
  end
  while *convergence criterion for π_d not met* **do**
    $\mathbb{D} = \{(\tau_i, c_{\tau_i})\}_{i=1}^N \leftarrow$`collect(`$d, N, E$`)`
    $\nabla_{\theta_d} J(\theta_d) \leftarrow$`compute-PG(`$\theta_d, \mathbb{D}$`)`
    $\theta_d \leftarrow$`optimizer.step(`$\theta_d, \nabla_{\theta_d} J(\theta_d)$`)`
  end
end

Procedure `collect(`d, N, E`)`

for $i : 1 \ldots N$ **do**
  $(s^i, g^i) \sim \rho_0$
  $s_1^i, \ldots, s_{2^d-1}^i \leftarrow$`predict-subgoals(`$s^i, g^i, d$`)`
  $\tau_i = [s^i, s_1^i, \ldots, s_{2^d-1}^i, g^i]$
  $c_{\tau_i} = [c_{j:j+1}]_{j=1}^{2^d-1} \leftarrow E$`.evaluate(`$\tau_i$`)`
end
return $\{(\tau_i, c_{\tau_i})\}_{i=1}^N$

Following the theoretical results in Section 4.1, where the greedy SGT trajectory used a different value function to predict sub-goals at different depths of the tree, we can expect a stochastic SGT policy that depends on the depth in the tree (d in Theorem 2) to perform better than a depth-independent policy. Theorem 2 holds true when π depends on d, similarly to a time-dependent policy in standard PG literature. We denote such a depth dependent policy as π_d, for $d \in 1, \ldots, D$. At depth $d = 0$, no sub-goal is predicted resulting in (s, g) being directly connected. Next, π_1 predicts a sub-goal s_m, segmenting the trajectory to two segments of depth 0: (s, s_m) and (s_m, g). The recursive construction continues, π_2 predicts a sub-goal and calls π_1 on the resulting segments, and so on until depth D.

An observation that we found important for improving training stability, is that the policies can be trained sequentially. Namely, we first train the d-depth policy, and only then start training $d+1$-depth policy, while freezing all policies of depth $\leq d$.

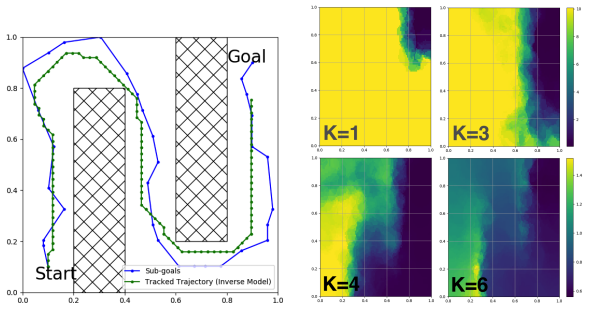
Our algorithm implementation, SGT-PG, is detailed in Algorithm 2. SGT-PG predicts sub-goals, and interacts with an environment E that evaluates the cost of segments (s, s'). SGT-PG maintains D depth-specific policies: $\{\pi_i\}_{i=1}^{D}$, each parametrized by $\{\theta_i\}_{i=1}^{D}$ respectively (i.e., we do not share parameters for policies at different depths, though this is possible in principle). The policies are trained in sequence, and every training cycle is comprised of on-policy data collection followed by policy update. The `collect` method collects on-policy data given N, the number of episodes to generate; d, the index of the policy being trained; and E, the environment. We found that for reducing noise when training π_d, it is best to sample from π_d only and take the mean predictions of $\{\pi_i\}_{i=1}^{d-1}$. The `compute-PG` method, based on Eq. (10), uses the collected data $\mathbb{D}$, and estimates $\nabla_{\theta_d} J(\theta_d)$. We found that similar to sequential RL, adding a trust region using the PPO optimization objective (Schulman et al., 2017) provides stable updates (see Section E.4 for specific loss function). Finally, `optimizer.step` updates θ_d according to any SGD optimizer algorithm, which completes a single training cycle for π_d. We proceed to train the next policy π_{d+1} when a pre-specified convergence criterion is met, for instance, until the expected cost of predicted trajectories $\mathbb{E}[c_\tau]$ stops decreasing.

6. Experiments

In this section we compare our *SGT* approach with the conventional *sequential* method (i.e. predicting the next state of the trajectory). We consider APSP problems inspired by robotic motion planning, where the goal is to find a collision-free trajectory between a pair of start and goal states. We illustrate the SGT value functions and trajectories on a simple 2D point robot domain, which we solve using Fitted SGTDP (Section 6.1, code: `https://github.com/tomjur/SGT_batch_RL.git`). We then consider a more challenging domain with a simulated 7DoF robotic arm, and demonstrate the effectiveness of SGT-PG (Section 6.2 code: `https://github.com/tomjur/SGT-PG.git`).

6.1. Fitted SGTDP Experiments

We start by evaluating the Fitted SGTDP algorithm. We consider a 2D particle moving in an environment with obstacles, as shown in Figure 2a. The particle can move a distance of 0.025 in one of the eight directions, and suffers a constant cost of 0.025 in free space, and a large cost of 10 on collisions. Its task is reaching from any starting point to within a 0.15 distance of any goal point without collision. This simple domain is a continuous-state optimal control Problem (2), and for distant start and goal points, as shown in Figure 2a, it requires relatively long-horizoned planning, making it suitable for studying batch RL algorithms.



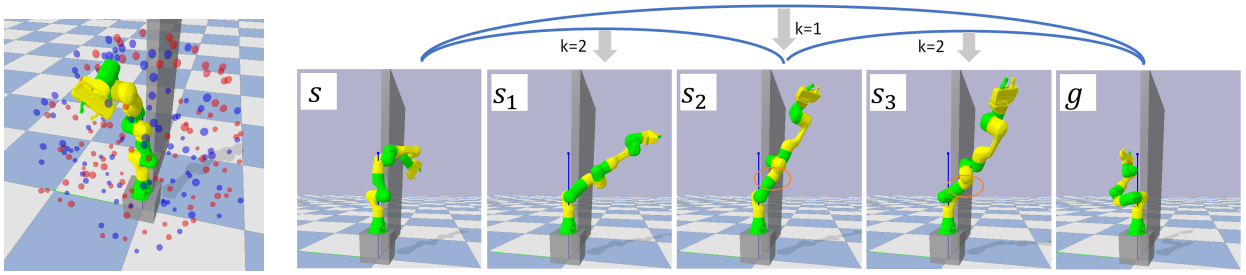
(a) (b)

Figure 2. Batch RL experiment. (a) A robot needs to navigate between the (hatched) obstacles. Blue - SGT prediction, green - trajectory tracking sub-goals using an inverse model. (b) Approximate values $\hat{V}_k(s, g = [0.9, 0.9])$ for several values of k. Note how the reachable region to the goal (non-yellow) grows with k.

To generate data, we sampled states and actions uniformly and independently, resulting in 125K (s, u, c, s') tuples. As for function approximation, we opted for simplicity, and used K-nearest neighbors (KNN) for all our experiments, with $\mathrm{K}_{neighbors} = 5$. To solve the minimization over states in Fitted SGTDP, we discretized the state space and searched over a 50×50 grid of points.

A natural baseline in this setting is FQI (Ernst et al., 2005; Riedmiller, 2005). We verified that for a fixed goal, FQI obtains near perfect results with our data. Then, to make it goal-conditioned, we used a universal Q-function (Schaul et al., 2015), requiring only a minor change in the algorithm (see supplementary for pseudo-code).

To evaluate the different methods, we randomly chose 200 start and goal points, and measured the distance from the goal the policies reach, and whether they collide with obstacles along the way. For FQI, we used the greedy policy with respect to the learned Q function. The approximate SGTDP method, however, does not automatically provide a policy, but only a state trajectory to follow. Thus, we experimented with two methods for extracting a policy from the learned sub-goal tree. The first is training an inverse model $f_{IM}(s, s')$ – a mapping from s, s' to u, using our data, and the same KNN function approximation. To reach a sub-goal g from state s, we simply run $f_{IM}(s, g)$ until we are close enough to g (we set the threshold to 0.15). An alternative method is first using FQI to learn a goal-based policy, as described above, and then running this policy on the sub-goals. The idea here is that the sub-goals learned by approximate SGTDP can help FQI overcome the long-horizon planning required in this task. Note that *all methods use exactly the same data, and the same function approximation* (KNN),



(a) (b)

Figure 3. Motion planning with *SGT-PG*. (a) Illustration of start (red) and goal (blue) positions of the end effector in *wall* domain. (b) SGT Trajectory. Note a non-trivial rotation between s_2 to s_3 (marked with orange circles) allowing linear motion from s_3 to g. In the supplementary material we also show a 7 sub-goals path.

	Avg. Distance to Goal	Avg. Collision Rate
approx. SGTDP + IM	0.13	0.25
approx. SGTDP + FQI	0.29	0.06
FQI	0.58	0.02

Table 1. Results for controllers of batch-RL experiments.

making for a fair comparison.

In Table 1 we report our results. FQI only succeed in reaching the very closest goals, resulting in a high average distance to goal. Fitted SGTDP (approx. SGTDP+IM), on the other hand, computed meaningful sub-goals for almost all test cases, resulting in a low average distance to goal when tracked by the inverse model. Figure 2 shows an example sub-goal tree and a corresponding tracked trajectory. The FQI policy did learn not to hit obstacles, resulting in the lowest collision rate. This is expected, as colliding leads to an immediate high cost, while the inverse model is not trained to take cost into account. Interestingly, combining the FQI policy with the sub-goals improves both long-horizon planning and short horizoned collision avoidance. In Figure 2b we plot the approximate value function $\hat{V}_k$ for different k and a specific goal at the top right corner. Note how the reachable parts of the state space to the goal expand with k.

6.2. Neural Motion Planning

An interesting application domain for our approach is neural motion planning (NMP, Qureshi et al., 2018) – learning to predict collision-free trajectories for a robot among obstacles. Here, we study NMP for the 7DoF Franka Panda robotic arm. Due to lack of space, full technical details of this section appear in supplementary Section E.1.

We follow an RL approach to NMP (Jurgenson & Tamar, 2019), using a cost function that incentivizes short, collision-free trajectories that reach the goal. The state space is the robot's 7 joint angles, and actions correspond to predicting the next state in the plan. Given the next predicted state, the robot moves by running a fixed PID controller in simulation, tracking a linear joint motion from current to next state, and a cost is incurred based on the resulting motion.

Model	# Sub-goals	*self-collision*	*wall*
SGT-PG	1	**1.±0.**	0.896±0.016
	3	**1.±0.**	**0.973±0.007**
	7	0.996±0.007	**0.973±0.007**
SeqSG	1	**1.±0.**	0.676±0.034
	3	0.983±0.007	0.593±0.047
	7	0.88±0.03	0.487±0.037

Table 2. Success rates for the NMP scenarios.

We formulate NMP as approximate APSP as follows. A model predicts $T-1$ sub-goals, resulting in T motion segments. Those segments are executed and evaluated *independently*, i.e. when evaluating segment (s, s') the robot first resets to s, and the resulting cost is based on its travel to s'.

Our experiments include two scenarios: *self-collision* and *wall*. In *self-collision*, there are no obstacles and the challenge is to generate a minimal distance path while avoiding self-collisions between the robot links. The more challenging *wall* workspace contains a wall that partitions the space in front of the robot (see Figure 3). In *wall*, the shortest path is often nonmyopic, requiring to first move *away* from the goal in order to pass the obstacle.

We compare *SGT-PG* with a sequential baseline, *Sequential sub-goals (SeqSG)*, which prescribes the sub-goals predictions *sequentially*. For appropriate comparison, both models use a PPO objective (Schulman et al., 2017), and a fixed architecture neural network to model the policy. All other hyper-parameters were specifically tuned for each model.

Table 2 compares *SGT-PG* and *SeqSG*, on predictions of 1, 3, and 7 sub-goals. We evaluate success rate (reaching goal without collision) on 100 random start-goal pairs that were held out during training (see Figure 3a). Each experiment was repeated 3 times and the mean and range are reported. Note that only 1 sub-goal is required to solve *self-collision*, and both models obtain perfect scores. For *wall*, on the other hand, more sub-goals are required, and here SGT

significantly outperforms *SeqSG*, which was not able to accurately predict several sub-goals.

7. Conclusion

We presented a framework for multi-goal RL that is derived from a novel first principle – the SGT dynamic programming equation. For deterministic domains, we showed that SGTs are less prone to drift due to approximation errors, reducing error accumulation from $O(N^2)$ to $O(N \log N)$. We further developed value-based and policy gradient RL algorithms for SGTs, and demonstrated that, in line with their theoretical advantages, SGTs demonstrate improved performance in practice.

Our work opens exciting directions for future research, including: (1) can our approach be extended to stochastic environments? (2) how to explore effectively based on SGTs? (3) can SGTs be extended to image-based tasks? Finally, we believe that our ideas will be important for robotics and autonomous driving applications, and other domains where goal-based predictions are important.

Acknowledgements

This work is partly funded by the Israel Science Foundation (ISF-759/19) and the Open Philanthropy Project Fund, an advised fund of Silicon Valley Community Foundation. Finally, the authors would like to thank Orr Krupnik for helpful comments.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bertsekas, D. Dynamic programming and optimal control: Volume II. 2005.

Bertsekas, D. P. and Tsitsiklis, J. N. *Neuro-dynamic programming*. Athena Scientific, 1996.

Bishop, C. M. Mixture density networks. Technical report, Citeseer, 1994.

Chan, T. M. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5): 2075–2089, 2010.

Deisenroth, M. P., Neumann, G., Peters, J., et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

Dhiman, V., Banerjee, S., Siskind, J. M., and Corso, J. J. Floyd-warshall reinforcement learning learning from past experiences to reach new goals. *arXiv preprint arXiv:1809.09318*, 2018.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Floyd, R. W. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

Fruit, R., Pirotta, M., Lazaric, A., and Brunskill, E. Regret minimization in mdps with options without prior knowledge. In *Advances in Neural Information Processing Systems*, pp. 3166–3176, 2017.

Ginsberg, M. L. Universal planning: An (almost) universally bad idea. *AI magazine*, 10(4):40–40, 1989.

Greensmith, E., Bartlett, P. L., and Baxter, J. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov): 1471–1530, 2004.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, 2017.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

Jayaraman, D., Ebert, F., Efros, A. A., and Levine, S. Time-agnostic prediction: Predicting predictable video frames. In *ICLR*, 2019.

Jurgenson, T. and Tamar, A. Harnessing reinforcement learning for neural motion planning. In *Robotics: Science and Systems (RSS)*, 2019.

Kaelbling, L. P. Goals as parallel program specifications. In *AAAI*, volume 88, pp. 60–65, 1988.

Kaelbling, L. P. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31 (3):360–375, 2012.

LaValle, S. M. *Planning algorithms*. Cambridge university press, 2006.

Mann, T. and Mannor, S. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *International conference on machine learning*, pp. 127–135, 2014.

McGovern, A. and Barto, A. G. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

Menache, I., Mannor, S., and Shimkin, N. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pp. 295–306. Springer, 2002.

Mishra, N., Abbeel, P., and Mordatch, I. Prediction and control with temporal segment models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2459–2468. JMLR. org, 2017.

Mülling, K., Kober, J., Kroemer, O., and Peters, J. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

Munos, R. and Szepesvári, C. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9 (May):815–857, 2008.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Pomerleau, D. A. ALVINN: An autonomous land vehicle in a neural network. In *NIPS*, pp. 305–313, 1989.

Qureshi, A. H., Bency, M. J., and Yip, M. C. Motion planning networks. *arXiv preprint arXiv:1806.05767*, 2018.

Riedmiller, M. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pp. 317–328. Springer, 2005.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.

Russell, S. J. and Norvig, P. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320, 2015.

Schoppers, M. J. *Representation and automatic synthesis of reaction plans*. PhD thesis, University of Illinois at Urbana-Champaign, 1989.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Şucan, I. A. and Kavraki, L. E. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pp. 449–464. Springer, 2009.

Şucan, I. A., Moll, M., and Kavraki, L. E. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. http://ompl.kavrakilab.org.

Sung, J., Jin, S. H., and Saxena, A. Robobarista: Object part based transfer of manipulation trajectories from crowdsourcing in 3d pointclouds. In *Robotics Research*, pp. 701–720. Springer, 2018.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.

Thorup, M. and Zwick, U. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.

Tsitsiklis, J. N. and Van Roy, B. Regression methods for pricing complex american-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

Williams, R. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pp. 664–673, 2014.

Zhang, H., Heiden, E., Julian, R., He, Z., Lim, J. J., and Sukhatme, G. S. Auto-conditioned recurrent mixture density networks for complex trajectory generation. *arXiv preprint arXiv:1810.00146*, 2018.