A. Proofs

Proof of Theorem 1

Proof. First, by definition, the shortest path from s to itself is 0. In the following, therefore, we assume that $s \neq s'$.

We show by induction that each $V_k(s, s')$ in Algorithm (4) is the cost of the shortest path from s to s' in 2^k steps or less.

Let $\tau_k(s, s')$ denote a shortest path from s to s' in 2^k steps or less, and let $V_k(s, s')$ denote its corresponding cost. Our induction hypothesis is that $V_{k-1}(s, s')$ is the cost of the shortest path from s to s' in 2^{k-1} steps or less. We will show that $V_k(s, s') = \min_{s_m} \{V_{k-1}(s, s_m) + V_{k-1}(s_m, s')\}$.

Assume by contradiction that there was some s^* such that $V_{k-1}(s, s^*) + V_{k-1}(s^*, s') < V_k(s, s')$. Then, the concatenated trajectory $[\tau_{k-1}(s, s^*), \tau_{k-1}(s^*, s')]$ would have 2^k steps or less, contradicting the fact that $\tau_k(s, s')$ is a shortest path from s to s' in 2^k steps or less. So we have that $V_k(s, s') \leq \{V_{k-1}(s, s_m) + V_{k-1}(s_m, s')\} \quad \forall s_m$. Since the graph is complete, $\tau_k(s, s')$ can be split into two trajectories of length 2^{k-1} steps or less. Let s_m be a midpoint in such a split. Then we have that $V_k(s, s') = \{V_{k-1}(s, s_m) + V_{k-1}(s_m, s')\}$. So equality can be obtained and thus we must have $V_k(s, s') = \min_{s_m} \{V_{k-1}(s, s_m) + V_{k-1}(s_m, s')\}$.

To complete the induction argument, we need to show that $V_0(s, s') = c(s, s')$. This holds since for $k = 0$, for each s, s', the only possible trajectory between them of length 1 is the edge s, s'.

Finally, since there are no negative cycles in the graph, for any s, s', the shortest path has at most N steps. Thus, for $k = \log_2(N)$, we have that $V_k(s, s')$ is the cost of the shortest path in N steps or less, which is the shortest path between s and s'. □

Proof of Proposition 1

Proof. The SGTDP Operator T is non-linear and not a contraction, but it is monotonic:

$$\forall s, g : V_\alpha(s, g) \leq V_\beta(s, g) \rightarrow \forall s, g : TV_\alpha(s, g) \leq TV_\beta(s, g). \tag{11}$$

To show (11), let $s'_m = \arg\min_{s_m} \{V_\beta(s, s_m) + V_\beta(s_m, g)\}$. Then:

$$\begin{aligned} TV_\alpha(s, g) &= \min_{s_m} \{V_\alpha(s, s_m) + V_\alpha(s_m, g)\} \leq V_\alpha(s, s'_m) + V_\alpha(s'_m, g) \\ &\leq V_\beta(s, s'_m) + V_\beta(s'_m, g) = \min_{s_m} \{V_\beta(s, s_m) + V_\beta(s_m, g)\} = TV_\beta(s, g) \end{aligned}$$

Back to the proof. By denoting $e = (1, 1, 1, 1, ..., 1)$ we can write (6) as:

$$V_0 - e\epsilon \leq \hat{V}_0 \leq V_0 + e\epsilon$$

Since T is monotonic, we can apply it on the inequalities:

$$T(V_0 - e\epsilon) \leq T\hat{V}_0 \leq T(V_0 + e\epsilon)$$

Focusing on the left-hand side expression (the right-hand side is symmetric) we obtain:

$$T(V_0 - e\epsilon)(s, s') = \min_{s_m} \{(V_0 - e\epsilon)(s, s_m) + (V_0 - e\epsilon)(s_m, s')\} = \min_{s_m} \{V_0(s, s_m) + V_0(s_m, s') - 2\epsilon\} = TV_0(s, s') - 2\epsilon$$

leading to:

$$TV_0 - 2e\epsilon \leq T\hat{V}_0 \leq TV_0 + 2e\epsilon$$

Using (6):

$$TV_0 - (2 + 1)e\epsilon \leq T\hat{V}_0 - e\epsilon \leq \hat{V}_1 \leq T\hat{V}_0 + e\epsilon \leq TV_0 + (2 + 1)e\epsilon$$

Proceeding similarly we obtain for $\hat{V}_2$:

$$T^2V_0 - [2(2+1)+1]e\epsilon \leq \hat{V}_2 \leq T^2V_0 + [2(2+1)+1]e\epsilon$$

And for every $k \geq 1$:

$$T^kV_0 - (2^{k+1}-1)e\epsilon \leq \hat{V}_k \leq T^kV_0 + (2^{k+1}-1)e\epsilon$$

$$\|\hat{V}_k - V_k\|_\infty = \|\hat{V}_k - T^kV_0\|_\infty \leq (2^{k+1}-1)\epsilon \tag{12}$$

For $k = \log_2 N$ we obtain:

$$\|\hat{V}_{\log_2 N} - V^*\|_\infty = \|\hat{V}_{\log_2 N} - V_{\log_2 N}\|_\infty \leq \epsilon(2N-1)$$

□

Proof of Proposition 2

Proof. For every iteration k, the middle state of any greedy SGT path with length 2^k is $\hat{s}_m = \arg\min_{s_m}\left\{\hat{V}_{k-1}(s,s_m) + \hat{V}_{k-1}(s_m,g)\right\}$. Thereby $\hat{s}_m$ fulfils the identity $\hat{V}_{k-1}(s,\hat{s}_m) + \hat{V}_{k-1}(\hat{s}_m,g) = T\hat{V}_{k-1}(s,g)$, for every iteration k, where T is the SGT operator. The next two relations then follow:

$$\begin{aligned}|\hat{V}_{k-1}(s,\hat{s}_m) + \hat{V}_{k-1}(\hat{s}_m,g) - V_k(s,g)| &= |T\hat{V}_{k-1}(s,g) - V_k(s,g)| \leq \\ &\leq |T\hat{V}_{k-1}(s,g) - \hat{V}_k(s,g)| + |\hat{V}_k(s,g) - V_k(s,g)| \leq \\ &\leq \epsilon + (2^{k+1}-1)\epsilon = 2^{k+1}\epsilon\end{aligned} \tag{13}$$

$$\begin{aligned}|V_{k-1}(s,\hat{s}_m) + V_{k-1}(\hat{s}_m,g) - V_k(s,g)| \leq &|\hat{V}_{k-1}(s,\hat{s}_m) + \hat{V}_{k-1}(\hat{s}_m,g) - V_k(s,g)| + \\ &+ |V_{k-1}(s,\hat{s}_m) - \hat{V}_{k-1}(s,\hat{s}_m)| + \\ &+ |V_{k-1}(\hat{s}_m,g) - \hat{V}_{k-1}(\hat{s}_m,g)| \leq \\ \leq &|\hat{V}_{k-1}(s,\hat{s}_m) + \hat{V}_{k-1}(\hat{s}_m,g) - V_k(s,g)| + 2\cdot(2^k-1)\epsilon\end{aligned} \tag{14}$$

Combining the two inequalities yields:

$$|V_{k-1}(s,\hat{s}_m) + V_{k-1}(\hat{s}_m,g) - V_k(s,g)| \leq (2^{k+1} + 2\cdot(2^k-1))\epsilon \leq 2^{k+2}\epsilon \tag{15}$$

Explicitly writing down relation (15) for all the different sub-paths we obtain:

$$\begin{array}{llll}
|V_{\log_2(N/2)}(s_0,s_{N/2}) & + V_{\log_2(N/2)}(s_{N/2},s_N) & - V_{\log_2(N)}(s_0,s_N) & | \leq 4N\epsilon \\
 & & & \\
|V_{\log_2(N/4)}(s_0,s_{N/4}) & + V_{\log_2(N/4)}(s_{N/4},s_{N/2}) & - V_{\log_2(N/2)}(s_0,s_{N/2}) & | \leq 2N\epsilon \\
|V_{\log_2(N/4)}(s_{N/2},s_{3N/4}) & + V_{\log_2(N/4)}(s_{3N/4},s_N) & - V_{\log_2(N/2)}(s_{N/2},s_N) & | \leq 2N\epsilon \\
 & & & \\
|V_{\log_2(N/8)}(s_0,s_{N/8}) & + V_{\log_2(N/8)}(s_{N/8},s_{N/4}) & - V_{\log_2(N/4)}(s_0,s_{N/4}) & | \leq N\epsilon \\
|V_{\log_2(N/8)}(s_{N/4},s_{3N/8}) & + V_{\log_2(N/8)}(s_{3N/8},s_{N/2}) & - V_{\log_2(N/4)}(s_{N/4},s_{N/2}) & | \leq N\epsilon \\
|V_{\log_2(N/8)}(s_{N/2},s_{5N/8}) & + V_{\log_2(N/8)}(s_{5N/8},s_{3N/4}) & - V_{\log_2(N/4)}(s_{N/2},s_{3N/4}) & | \leq N\epsilon \\
|V_{\log_2(N/8)}(s_{3N/4},s_{7N/8}) & + V_{\log_2(N/8)}(s_{7N/8},s_N) & - V_{\log_2(N/4)}(s_{3N/4},s_N) & | \leq N\epsilon \\
 & \vdots & & \\
|V_0(s_0,s_1) & + V_0(s_1,s_2) & - V_1(s_0,s_2) & | \leq 8\epsilon \\
|V_0(s_2,s_3) & + V_0(s_3,s_4) & - V_1(s_2,s_4) & | \leq 8\epsilon
\end{array}$$

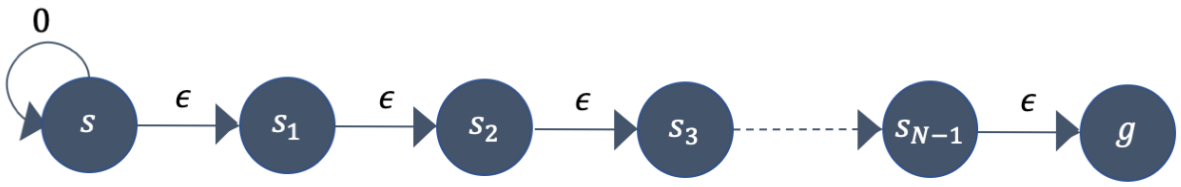


Figure 4. Example graph with a start and goal states s, g where the optimal path has a total cost of $N\epsilon$ while ϵ-approximated Bellman might form a path with total cost of infinity

$$\vdots$$

$$|V_0(s_{N-2}, s_{N-1}) \quad + V_0(s_{N-1}, s_N) \quad - V_1(s_{N-2}, s_N) \quad | \leq 8\epsilon$$

Summing all the inequalities, along with the triangle inequality, lead to the following:

$$|\sum_{i=0}^{N-1} V_0(s_i, s_{i+1}) - V_{\log_2(N)}(s_0, s_N)| \leq 4N \log_2(N)\epsilon$$

The identities $c(s_i, s_{i+1}) = V_0(s_i, s_{i+1})$ and $V^*(s_0, s_N) = V_{\log_2 N}(s_0, s_N)$ complete the proof. □

Proposition 2 provides a bound on the approximate shortest path using SGTDP. In contrast, we show that a similar bound for the sequential Bellman approach does not hold. For a start and goal pair s, g, and an approximate Bellman value function $\hat{V}^B$, let $s_0^B, \ldots, s_N^B$ denote the greedy shortest path according to the Bellman operator, i.e., $s_0 = s, s_N = g$ and for $1 \leq k < N$: $s_{k+1} = \arg\min_{s_m} \left\{ c(s_k, s_m) + \hat{V}^B(s_m, g) \right\}$. Note that the greedy Bellman update is not guaranteed to generate a trajectory that reaches the goal, therefore we add g as the last state in the trajectory, following our convention that the graph is fully connected. When evaluating the greedy trajectory error in the sequential approach, we deal with two different implementation cases: Using one approximated value function or N approximated value functions. The next proposition shows that when using one value function the greedy trajectory can be arbitrarily bad, regardless of ϵ. Propositions (4) and (5) show that the error of the greedy trajectory, using N value functions, have a tight $\mathcal{O}(N^2)$ bound.

Proposition 3. *For any ϵ, β, there exists a graph and an approximate value function $\hat{V}^B$ satisfying $\|\hat{V}^B - V^*\|_\infty \leq \epsilon N$, such that $\sum_{i=0}^{N-1} c(s_i, s_{i+1}) \geq V^*(s, g) + \beta$.*

Proof. We show a graph example (Figure 4), where a ϵ-approximate Bellman value function $\hat{V}^B$ might form the path $s_0, s_0, \ldots, s_0, g$ with a total cost of infinity. (Reminder: The graph is fully connected, all the non-drawn edges have infinity cost), while the optimal path $s_0, s_1, \ldots, s_{N-1}, g$ has a total cost of $N\epsilon$.
As $\hat{V}^B$ is independent of the current time-step k, and the optimal values for s and s_1 are $N\epsilon$ and $(N-1)\epsilon$ respectively, due to approximation error $\hat{V}^B$ (for every time-step) might suggest that the value of s_0 is lower than the value of s_1: $\hat{V}^B(s_0, g) - N\epsilon \leq \hat{V}^B(s_1, g) + N\epsilon$. The resulting policy will choose to stay in s_0 for the first $N-1$ steps. Since the maximum path length is N and the path must end at the goal, the last step will be directly from s_0 to g, resulting in a cost of infinity.

□

Proposition 4. *For a finite state graph, start and goal pair s, g, and sequence of ϵ-approximate Bellman value functions $\hat{V}_0^B, \ldots, \hat{V}_N^B$ satisfying $\|\hat{V}_{k+1}^B - T^B \hat{V}_k^B\|_\infty \leq \epsilon$ and $\|\hat{V}_0^B - V_0\|_\infty \leq \epsilon$, let $s_0, \ldots, s_N$ denote the greedy Bellman path, that is: $s_0 = s, s_N = g, s_{k+1} = \arg\min_{s_{k+1}} c(s_k, s_{k+1}) + \hat{V}_{N-k-2}^B(s_{k+1}, g)$, etc. We have that $\sum_{i=0}^{N-1} c(s_i, s_{i+1}) \leq V^*(s, g) + (N^2 - N)\epsilon = V^*(s, g) + \mathcal{O}(N^2)$.*

Proof. Denote V_k as the Bellman value function at iteration k, $\hat{V}_k$ as the ϵ-approximated Bellman value function and T_B as the Bellman operator. Using the properties $\|T_B \hat{V}_{k-1} - \hat{V}_k\|_\infty \leq \epsilon$ and $\|V_k - \hat{V}_k\|_\infty \leq k\epsilon$ (Bertsekas & Tsitsiklis, 1996, Pg. 332), the following relation holds:

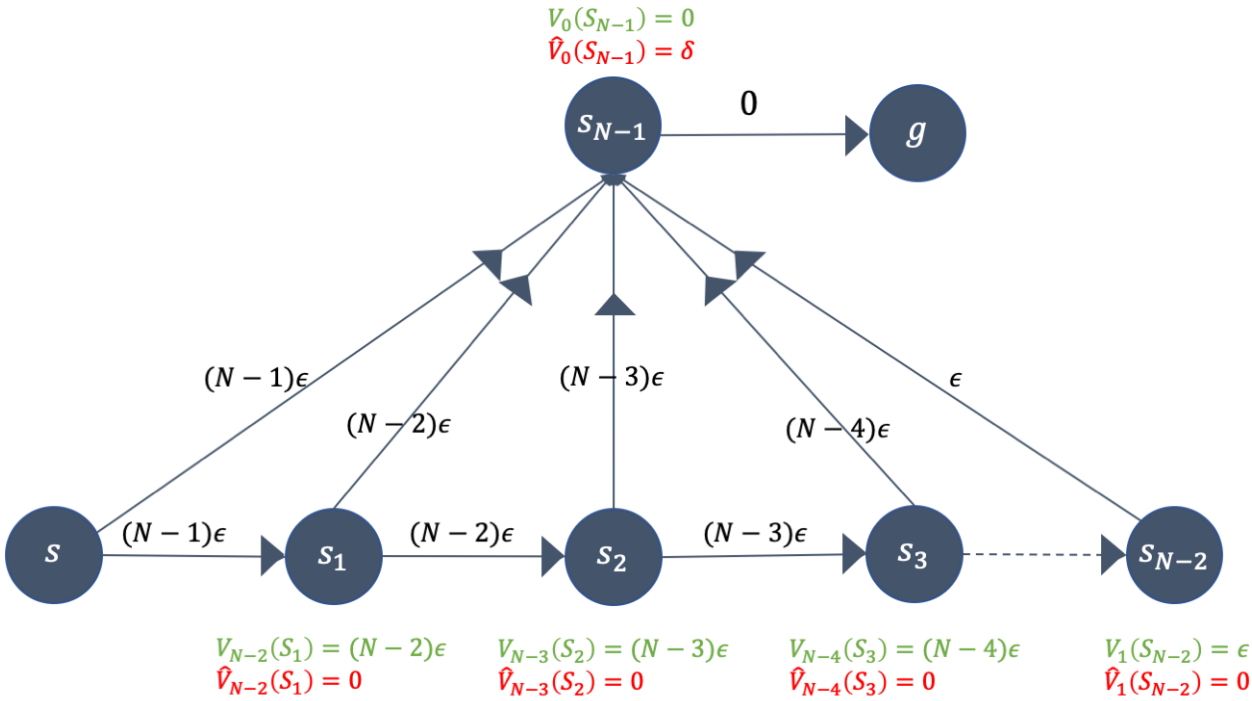


Figure 5. Example graph with a start and goal states s, g. The edges costs are marked in black labels, the Bellman value function at iteration k is marked in green and the ϵ-approximated Bellman value is marked in red. The optimal path has a total cost of $V^*(s,g) = (N-1)\epsilon$ while the ϵ-approximated Bellman might form the path going through all the states with a total cost of $N^2 - N\epsilon/2 = V^*(s,g) + \mathcal{O}(N^2)$.

for $0 \leq k \leq N-2$:

$$\begin{aligned} c(s_k, s_{k+1}) &= c(s_k, s_{k+1}) + \hat{V}_{N-k-2}(s_{k+1}, g) - \hat{V}_{N-k-2}(s_{k+1}, g) \\ &= (T_B \hat{V}_{N-k-2})(s_k, g) - \hat{V}_{N-k-2}(s_{k+1}, g) \\ &\leq (\hat{V}_{N-k-1}(s_k, g) + \epsilon) - \hat{V}_{N-k-2}(s_{k+1}, g) \\ &\leq V_{N-k-1}(s_k, g) + (N-k-1)\epsilon + \epsilon - V_{N-k-2}(s_{k+1}, g) + (N-k-2)\epsilon \\ &\leq V_{N-k-1}(s_k, g) - V_{N-k-2}(s_{k+1}, g) + 2(N-k-1)\epsilon \end{aligned}$$

for $k = N-1$:

$$c(s_k, s_{k+1}) = c(s_{N-1}, g) = V_0(s_{N-1}, g)$$

Summing all the path costs:

$$\begin{aligned} \sum_{k=0}^{N-1} c(s_k, s_{k+1}) &= V_0(s_{N-1}, g) + \sum_{k=0}^{N-2} c(s_k, s_{k+1}) \\ &\leq V_0(s_{N-1}, g) + \sum_{k=0}^{N-2} V_{N-k-1}(s_k, g) - V_{N-k-2}(s_{k+1}, g) + 2(N-k-1)\epsilon \\ &= V_0(s_{N-1}, g) + V_{N-1}(s_0, g) - V_0(s_{N-1}, g) + (N^2 - N)\epsilon = V^*(s_0, g) + (N^2 - N)\epsilon \end{aligned}$$

□

Proposition 5. *The error bound stated in Proposition(4), $\sum_{i=0}^{N-1} c(s_i, s_{i+1}) - V^*(s,g) \leq \mathcal{O}(N^2)$, is indeed a tight bound.*

Proof. We show a graph example (Figure 5),where a sequence of ϵ-approximate Bellman value functions $\hat{V}_0, \ldots, \hat{V}_{N-1}$ might form the path from the start to the goal state, going through all the available states, with a total cost of $(N^2 - N)/2 = \mathcal{O}(N^2)$, while the optimal path (s_0, s_{N-1}, s_N) has a total cost of $V^*(s, g) = (N-1)\epsilon$.
The Bellman value of every state s_k, $0 \leq k \leq s_{N-3}$: $V_{N-k-1}(s_k, g) = (N-k-1)\epsilon$, where $N-k-1$ is the time horizon from state s_k to the goal. Since the ϵ-approximate Bellman value function at every state s_k might have a maximum approximation error of $(N-k-1)\epsilon$, it might suggest that $\hat{V}_{N-k-1}(s_k, g) = 0$ for every $1 \leq k \leq N-2$. The Bellman value for state s_{N-1} is 0 and the ϵ-approximate Bellman value might be δ for some $0 < \delta \leq \epsilon$. The greedy Bellman policy has to determine at every state s_k, $0 \leq k \leq s_{N-3}$, whether to go to s_{k+1} or to s_{N-1}. Since both possible actions have an immediate cost of $(N-k-1)\epsilon$, the decision will only be based on the evaluation of s_{k+1} and s_{N-1} with the ϵ-approximate Bellman value function: $\hat{V}_{N-k-1}(s_k, g) = 0 < \delta = \hat{V}_0(s_{N-1}, g)$. Therefore, the greedy bellman policy will decide at every state s_k to go to s_{k+1}, forming the path with a total cost of $\sum_{k=1}^{N-1} k\epsilon = N^2 - N\epsilon/2$

□

B. Policy Gradient Theorem for Sub-goal Trees

In this section we provide the mathematical framework and tool we used in the Policy Gradient Theorem 2 in Section 5.2 which allows us to formulate the SGT-PG algorithm (Section 5.2). The SGT the prediction process (Eq.(8)) no-longer decomposes sequentially like a MDP, therefore, we provide a different mathematical construct:

Finite-depth Markovian sub-goal tree (FD-MSGT) is a process predicting sub-goals (or intermediate states) of a trajectory in a dynamical system as described in (3). The process evolves trajectories recursively (as we describe in detail below), inducing a tree-like decomposition for the probability of a trajectory as described Eq. (8). In this work we consider trees with fixed levels D (corresponding to finite-horizon MDP in sequential prediction RL with horizon $T = 2^D$) [7]. Formally, a FD-MSGT is comprised of (S, ρ_0, c, D) where S is the state space, ρ_0 is the initial start-goal pairs distribution, c is a non-negative cost function obeying Eq. (3), and D is the depth of the tree [8].

Recursive evolution of FD-MSGT: Formally, an initial pair $(s_0 = s, s_T = g) \sim \rho_0$ is sampled, defining the root of the tree. Next, a policy $\pi(s_{\frac{T}{2}}|s_0, s_T)$ is used to predict the sub-goal $s_{\frac{T}{2}}$, creating two new tree nodes of depth D corresponding to the segments $(s_0, s_{\frac{T}{2}})$ and $(s_{\frac{T}{2}}, s_T)$. Recursively each segment is again partitioned using π resulting in four tree nodes of level $D-1$ corresponding to segments $(s_0, s_{\frac{T}{4}})$, $(s_{\frac{T}{4}}, s_{\frac{T}{2}})$, $(s_{\frac{T}{2}}, s_{\frac{3T}{4}})$ and $(s_{\frac{3T}{4}}, s_T)$. The process continues recursively until the depth of the tree is 1. This results in Eq.(8), namely,

$$\Pr_\pi[\tau|s,g] = \Pr_\pi[s_0,\ldots,s_T|s,g] = \Pr_\pi[s_0,\ldots,s_{\frac{T}{2}}|s,s_m]\Pr_\pi[s_{\frac{T}{2}},\ldots,s_T|s_m,g]\pi(s_m|s,g).$$

FD-MSGT results in $\sum_{d=0}^{D-1} 2^d = 2^D - 1$ sub-goals, and $2^D + 1$ overall states (including s and g), setting $T = 2^D$. Finally, c defines the cost of the prediction $c(\tau) = c_{0:T}$, and is evaluated on the leaves of the FD-MSGT tree according to Eq. (3). Figure 3b illustrates a FD-MSGT of $D = 2$. The objective of FD-MSGT is to find a policy $\pi : S^2 \to S$ minimizing the expected cost of a trajectory $J^\pi = \mathbb{E}_{\tau\sim\rho(\pi)}[c_\tau]$.

Non-recursive formulation for trajectory likelihood: We next derive a non-recursive formulation of Eq. (8), using the notations defined in Theorem 2, namely, $s^{i,d} = s_{(i-1)\cdot 2^d}$, $s_m^{i,d} = s_{(2i-1)\cdot 2^{d-1}}$, $g^{i,d} = s_{i\cdot 2^d}$, 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. We re-arrange the terms in (8) grouping by depth resulting in a non-recursive formula,

$$\Pr_\pi[\tau|s,g] = \Pr_\pi[s_0,\ldots s_T|s,g] = \prod_{d=1}^{D}\prod_{i=1}^{2^{D-d}} \pi\left(s_m^{i,d}\middle| s^{i,d}, g^{i,d}\right). \tag{16}$$

For example consider a FD-MSGT with $D = 3$ ($T = 8$). Given $s_0 = s$ and $s_8 = g$, the sub-goals will be $s_1, \ldots s_7$. We enumerate the two products in Eq. (16) resulting in:

1. $D = 3$**:** Results in a single iteration of $i = 1 = 2^{3-3}$.
$$s^{1,3} = s_{(1-1)\cdot 2^3} = s_0, \quad s_m^{1,3} = s_{(2\cdot 1-1)\cdot 2^{3-1}} = s_4, \quad g^{1,3} = s_{1\cdot 2^3} = s_8$$

2. $D = 2$**:** In this case $i \in [1, 2^{3-2} = 2]$.
$$s^{1,2} = s_{(1-1)\cdot 2^2} = s_0, \quad s_m^{1,2} = s_{(2\cdot 1-1)\cdot 2^{2-1}} = s_2, \quad g^{1,2} = s_{1\cdot 2^2} = s_4$$
$$s^{2,2} = s_{(2-1)\cdot 2^2} = s_4, \quad s_m^{2,2} = s_{(2\cdot 2-1)\cdot 2^{2-1}} = s_6, \quad g^{2,2} = s_{2\cdot 2^2} = s_8$$

3. $D = 1$**:** Finally, predicts the leaves resulting in $i \in [1, 2^{3-1} = 4]$.
$$s^{1,1} = s_{(1-1)\cdot 2^1} = s_0, \quad s_m^{1,1} = s_{(2\cdot 1-1)\cdot 2^{1-1}} = s_1, \quad g^{1,1} = s_{1\cdot 2^1} = s_2$$
$$s^{2,1} = s_{(2-1)\cdot 2^1} = s_2, \quad s_m^{2,1} = s_{(2\cdot 2-1)\cdot 2^{1-1}} = s_3, \quad g^{2,1} = s_{2\cdot 2^1} = s_4$$
$$s^{3,1} = s_{(3-1)\cdot 2^1} = s_4, \quad s_m^{3,1} = s_{(2\cdot 3-1)\cdot 2^{1-1}} = s_5, \quad g^{3,1} = s_{3\cdot 2^1} = s_6$$
$$s^{4,1} = s_{(4-1)\cdot 2^1} = s_6, \quad s_m^{4,1} = s_{(2\cdot 4-1)\cdot 2^{1-1}} = s_7, \quad g^{4,1} = s_{4\cdot 2^1} = s_8$$

[7] Extending MSGT to infinite recursion without a depth restrictions (similar to infinite-horizon MDP) is left for future work.

[8] Note the crucial difference between MSGTs and MDPs. MSGTs operate on pairs of states from S, whereas MDPs, which are usually not goal conditioned, operate on single states.

Allowing us to assert the equivalence of the explicit and recursive formulations in this case.

B.1. Proof of Theorem 2

Let π_θ be a policy with parameters θ, we next prove a policy gradient theorem (Theorem 3) for computing $\nabla_\theta J^{\pi_\theta}$, using the following proposition:

Proposition 6. *Let π_θ be a policy with parameters θ of a FD-MSGT with depth D, then*

$$\nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau] = \sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} \nabla_\theta \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right) \tag{17}$$

Proof. First, we express $\Pr_{\rho(\pi_\theta)}[\tau]$ using Eq. 16 and ρ_0 obtaining,

$$\Pr_{\rho(\pi_\theta)}[\tau] = \rho(s_0, s_T) \cdot \prod_{d=1}^{D} \prod_{i=1}^{2^{D-d}} \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right).$$

Next, by taking the log we have,

$$\log \Pr_{\rho(\pi_\theta)}[\tau] = \log \rho(s_0, s_T) + \sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right).$$

and taking the gradient w.r.t θ yields Eq. (17) □

Proposition 6 shows that the gradient of a trajectory w.r.t. θ does not depend on the initial distribution. This allows us to derive a policy gradient theorem:

Theorem 3. *Let π_θ be a stochastic SGT policy, $\rho(\pi_\theta)$ be a trajectory distribution defined above, and $T = 2^D$. Then*

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

Proof. To obtain $\nabla_\theta J(\theta)$ we write Eq. (9) as an explicit expectation and use $\nabla_x f(x) = f(x) \cdot \nabla_x \log f(x)$:

$$\nabla_\theta J(\theta) = \sum_\tau c(\tau) \cdot \nabla_\theta \Pr_{\rho(\pi_\theta)}[\tau] = \sum_\tau c(\tau) \cdot \Pr_{\rho(\pi_\theta)}[\tau] \cdot \nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau] = \mathbb{E}_{\rho(\pi_\theta)} \left[c(\tau) \nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau] \right]$$

This proves the first equality. For the second equality we substitute $\nabla_\theta \log \Pr_{\rho(\pi_\theta)}[\tau]$ according to Eq. (17). □

The policy gradient theorem allows estimating $\nabla J(\theta)$ from on policy data collected using π_θ. We next show how to improve the estimate in Theorem 3 using control variates (baselines). We start with the following baseline-reduction proposition.

Proposition 7. *Let π_θ be a policy with parameters θ of a FD-MSGT with depth D, and let $b^{i,d} = b(s^{i,d}, g^{i,d})$ be any fixed function $b^{i,d} : S^2 \to R$, then*

$$\mathbb{E}_{\rho(\pi_\theta)} \left[\sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} b^{i,d} \cdot \nabla_\theta \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right) \right] = 0. \tag{18}$$

Proof. First we establish a useful property. Let $p_\theta(z)$ be some parametrized distribution. Differentiating $\sum_z p_\theta(z) = 1$ yields

$$\sum\nolimits_z (\nabla \log p_\theta(z)) p_\theta(z) = \mathbb{E}^\theta(\nabla \log p_\theta(z)) = 0. \tag{19}$$

Then we expand the left-hand side of Eq. (18), and use Eq. (19) in the last row:

$$\begin{aligned}
\mathbb{E}_{\rho(\pi_\theta)}\left[\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}} b^{i,d}\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\right] &= \sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[b^{i,d}\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\right] = \\
&\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[\mathbb{E}_{\rho(\pi_\theta)}\left[b^{i,d}\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s^{i,d},g^{i,d}\right]\right] = \\
&\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[b^{i,d}\cdot\mathbb{E}_{\rho(\pi_\theta)}\left[\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s^{i,d},g^{i,d}\right]\right] = \\
&\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[b^{i,d}\cdot 0\right] = 0,
\end{aligned}$$

which concludes the proof.

□

We define $b : S^2 \to R$ as a *segment-dependent baseline* if b is a fixed function that operates on a pair of states s and g. The last proposition allows us to reduce segment-dependent baselines, $b^{i,d}$, from the estimations of $\nabla J(\theta)$ without bias. Finally, in the next proposition we show that instead of estimating $\nabla J(\theta)$ using c_τ we can instead use $C^{i,d}$ as follows:

Proposition 8. *Let π_θ be a policy with parameters θ of a FD-MSGT with depth D, then:*

$$\nabla_\theta J(\theta) = \sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[c_{(i-1)\cdot 2^d:i\cdot 2^d}\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\right]$$

Proof. We have that:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \mathbb{E}_{\rho(\pi_\theta)}\left[c_\tau\cdot\sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\right] = \sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[c_\tau\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\right] \\
=& \sum_{d=1}^{D}\sum_{i=1}^{2^{D-d}}\mathbb{E}_{\rho(\pi_\theta)}\left[\mathbb{E}_{\rho(\pi_\theta)}\left[c_\tau\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s_0\dots s^{i,d},g^{i,d}\dots s_T\right]\right]. \qquad (20)
\end{aligned}$$

The first transition is the expectation of sums, and the second is the smoothing theorem. We next expand the inner expectation in Eq. (20), and partition the sum of cost c_τ into three sums: between indices 0 to $(i-1)\cdot 2^d$, $(i-1)\cdot 2^d$ to $i\cdot 2^d$, and finally from $i\cdot 2^d$ to T.

$$\begin{aligned}
&\mathbb{E}_{\rho(\pi_\theta)}\left[c_\tau\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s_0\dots s^{i,d},g^{i,d}\dots s_T\right] \\
&= \mathbb{E}_{\rho(\pi_\theta)}\left[\left(c_{0:(i-1)\cdot 2^d}+c_{i\cdot 2^d:T}\right)\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s_0\dots s^{i,d},g^{i,d}\dots s_T\right] \\
&+ \mathbb{E}_{\rho(\pi_\theta)}\left[c_{(i-1)\cdot 2^d:i\cdot 2^d}\cdot\nabla_\theta\log\pi_\theta\left(s_m^{i,d}\middle|s^{i,d},g^{i,d}\right)\middle|s_0\dots s^{i,d},g^{i,d}\dots s_T\right].
\end{aligned}$$

Next, by the definition of FD-MSGT the costs between s_0 to $s_{(i-1)\cdot 2^d}$ and $s_{i\cdot 2^d}$ to s_T, are independent of the policy predictions between indices $(i-1)\cdot 2^d$ to $i\cdot 2^d$ – making them constants. As we showed in Proposition 7, we obtain that

this expectation is 0. Thus the expression for $\nabla_\theta J(\theta)$ simplifies to:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} \mathbb{E}_{\rho(\pi_\theta)} \left[\mathbb{E}_{\rho(\pi_\theta)} \left[c_\tau \cdot \nabla_\theta \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right) \middle| s_0 \dots s^{i,d}, g^{i,d} \dots s_T \right] \right] \\
&= \sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} \mathbb{E}_{\rho(\pi_\theta)} \left[\mathbb{E}_{\rho(\pi_\theta)} \left[c_{(i-1)\cdot 2^d : i \cdot 2^d} \cdot \nabla_\theta \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right) \middle| s_0 \dots s^{i,d}, g^{i,d} \dots s_T \right] \right] \\
&= \sum_{d=1}^{D} \sum_{i=1}^{2^{D-d}} \mathbb{E}_{\rho(\pi_\theta)} \left[c_{(i-1)\cdot 2^d : i \cdot 2^d} \cdot \nabla_\theta \log \pi_\theta \left(s_m^{i,d} \middle| s^{i,d}, g^{i,d} \right) \right] \qquad (21)
\end{aligned}$$

□

Finally, combining Propositions 7 and 8 in Theorem 3 provides us Theorem 2 in the main text.

C. Batch RL Baselines

In Algorithm 3 we present a goal-based versions of fitted-Q iteration (FQI) (Ernst et al., 2005) using universal function approximation (Schaul et al., 2015), which we used as a baseline in our experiments.

Algorithm 3 Fitted Q with Universal Function Approximation

Algorithm

Input: dataset $D = \{s, u, c, s'\}$, Goal reached threshold δ

Create transition data set $D_{trans} = \{s, u, s'\}$ and targets $T_{trans} = \{c\}$ with s, s', c taken from D

Fit $\hat{Q}(s, u, s')$ to data in D_{trans}

for $k : 1...K$ **do**

Create random goal data set $D_{goal} = \{s, u, g\}$ and targets

$$T_{goal} = \left\{ \left\{ c(s, u) + \min_{u'} \hat{Q}(s', u', g)_{||s'-g||>\delta} \right\} \right\}$$

with s, u, s' taken from D and g randomly chosen from states in D

Fit $\hat{Q}(s, u, s')$ to data in D_{goal}

end

Next, in Algorithm 4 we present an approximate dynamic programming version of Floyd-Warshall RL (Kaelbling, 1993) that corresponds to the batch RL setting we investigate. This algorithm was not stable in our experiments, as the value function converged to zero for all states (when removing the self transition fitting in line 7 of the algorithm, the values converged to a constant value).

Algorithm 4 Approximate Floyd Warshall

Algorithm

Input: dataset $D = \{s, u, c, s'\}$, Maximum path cost C_{max}

Create transition data set $D_{trans} = \{s, s'\}$ and targets $T_{trans} = \{c\}$ with s, s', c taken from D

Create random transition data set $D_{random} = \{s, s_{rand}\}$ and targets $T_{random} = \{C_{max}\}$ with s, s_{rand} randomly chosen from states in D

Create self transition data set $D_{self} = \{s, s\}$ and targets $T_{self} = \{0\}$ with s taken from D

Fit $\hat{V}(s, s')$ to data in $D_{trans}, D_{random}, D_{self}$

for $k : 1...K$ **do**

Create random goal and mid-point data set $D_{goal} = \{s, g\}$ and targets

$$T_{goal} = \left\{ \min \left\{ \hat{V}(s, g), \hat{V}(s, s_m) + \hat{V}(s_m, g) \right\} \right\}$$

with s, s_m, g randomly chosen from states in D

Create self transition data set $D_{self} = \{s, s\}$ and targets $T_{self} = \{0\}$ with s taken from D

Fit $\hat{V}(s, s')$ to data in D_{goal}, D_{self}

end

D. Learning SGT with Supervised Learning

In this section we describe how to learn SGT policies for deterministic, stationary, and finite time dynamical systems from labeled data generated by an expert. This setting is commonly referred to as Imitation Learning (IL). Specifically, we are given a dataset $D^{\pi^*} = \{\tau_i\}_{i=1}^N$ of N trajectory demonstrations, provided by an expert policy π^*. Each trajectory demonstration τ_i from the dataset D^{π^*}, is a sequence of T_i states, i.e. $\tau_i = s_0^i, s_1^i \dots s_{T_i}^i$.[9] In this work, we assume a goal-based setting, that is, we assume that the expert policy generates trajectories that lead to a goal state which is the last state in each trajectory demonstration. Our goal is to learn a policy that, given a pair of current state and goal state (s, g), predicts the trajectory that π^* would have chosen to reach g from s.

D.1. Imitation learning with SGT

In this section, we describe the learning objectives for *sequential* and *sub-goal tree* prediction approaches under the IL settings. We focus on the Behavioral Cloning (BC)(Pomerleau, 1989) approach to IL, where a parametric model for a policy $\hat{\pi}$ with parameters θ is learned by maximizing the log-likelihood of observed trajectories in D^{π^*}, i.e.,

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau_i \sim D^{\pi^*}} \left[\log P_{\hat{\pi}}(\tau_i = s_0^i, s_1^i, \dots s_T^i | s_0^i, s_T^i; \theta)\right]. \tag{22}$$

Denote the horizon T as the maximal number of states in a trajectory. For ease of notation we assume T to be the same for all trajectories[10]. Also, let $s_i = s_0^i$ and $g_i = s_{T_i}^i$ denote the start and goal states for τ_i. We ask – how to best represent the distribution $P_{\hat{\pi}}(\tau | s, g; \theta)$?

The *sequential* trajectory representation (Pomerleau, 1989; Ross et al., 2011; Zhang et al., 2018; Qureshi et al., 2018), a popular approach, decomposes $P_{\hat{\pi}}(s_0, s_1, \dots s_T | s, g; \theta)$ by sequentially predicting states in the trajectory conditioned on previous predictions. Concretely, let $h_t = s_0, s_1, \dots, s_t$ denote the history of the trajectory at time index t, the decomposition assumed by the *sequential* representation is $P_{\hat{\pi}}(s_0, s_1, \dots s_T | s, g; \theta) = P_{\hat{\pi}}(s_1 | h_0, g; \theta) P_{\hat{\pi}}(s_2 | h_1, g; \theta) \dots P_{\hat{\pi}}(s_T | h_{T-1}, g; \theta)$. Using this decomposition, (22) becomes:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau_i \sim D^{\pi^*}} \left[\sum_{t=1}^{T} \log P_{\hat{\pi}}(s_{t+1}^i | h_t^i, g^i; \theta)\right]. \tag{23}$$

We can learn $P_{\hat{\pi}}$ using a batch stochastic gradient descent (SGD) method. To generate a sample (s_t, h_{t-1}, g) in a training batch, a trajectory τ_i is sampled from D^{π^*}, and an observed state s_t^i is further sampled from τ_i. Next, the history h_{t-1}^i and goal g_i are extracted from τ_i. After learning, sampling a trajectory between s and g is a straight-forward iterative process, where the first prediction is given by $s_1 \sim P_{\hat{\pi}}(s | h_0 = s, g; \theta)$, and every subsequent prediction is given by $s_{t+1} \sim P_{\hat{\pi}}(s | h_t = (s, s_1, \dots s_t), g; \theta)$. This iterative process stops once some stopping condition is met (such as a target prediction horizon is reached, or a prediction is 'close-enough' to g).

The Sub-Goal Tree Representation: In SGT we instead predict the middle-state in a divide-and-conquer approach. Following Eq. (8) we can redefine the maximum-likelihood objective as

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau_i \sim D^{\pi^*}} \Big[\log P_{\hat{\pi}}(s_{T/2}^i | s^i, g^i; \theta) + \log P_{\hat{\pi}}(s_{T/4}^i | s^i, s_{T/2}^i; \theta) + \log P_{\hat{\pi}}(s_{3T/4}^i | s_{T/2}^i, g^i; \theta) \dots \Big]. \tag{24}$$

To organize our data for optimizing Eq. (24), we first sample a trajectory τ_i from D^{π^*} for each sample in the batch. From τ_i we sample two states s_a^i and s_b^i and obtain their midpoint $s_{\frac{a+b}{2}}^i$. Pseudo-code for sub-goal tree prediction and learning are provided in Algorithm 5 and Algorithm 6.

D.2. Imitation Learning Experiments

We compare the *sequential* and *Sub-Goal Tree* (SGT) approaches for BC. Consider a point-robot motion-planning problem in a 2D world, with two obstacle scenarios termed *simple* and *hard*, as shown in Figure 6. In *simple*, the distribution of possible motions from left to right is uni-modal, while in *hard*, at least 4 modalities are possible.

[9] We note that limiting our discussion to states only can be easily extended to include actions as well by concatenating states and actions. However, we refrain from that in this work in order to simplify the notations, and remain consistent with the main text.

[10] For trajectories with $T_i < T$ assume $s_i^{T_i}$ repeats $T - T_i$ times, alternatively, generate middle states from data until $T_i = T$

Algorithm 5 Sub-Goal Tree BC Trajectory Prediction

Algorithm

Input: parameters θ of parametric distribution $P_{\hat{\pi}}$, start state s, goal state g, max depth K

return $[s]$ + `PredictSGT(`θ, s, g, K`)` +$[g]$

Procedure `PredictSGT(`θ, s_1, s_2, k`)`

if $k > 0$ **then**

Predict midpoint $s_m \sim P_{\hat{\pi}}(s_m|s_1, s_2; \theta)$

return `PredictSGT(`θ, s_1, s_m, $k-1$`)` + $[s_m]$ + `PredictSGT(`θ, s_m, s_2, $k-1$`)`

end

Algorithm 6 Sub-Goal Tree BC SGD-Based Training

Algorithm

Input: dataset $D = \left\{\tau_i = s_0^i, s_1^i \dots s_{T_i}^i\right\}_{i=1}^N$, train steps M, batch size B

Initialize parameters θ for parametric distribution $P_{\hat{\pi}}(s_m|s, g; \theta)$

for $i : 1...M$ **do**

Sample batch of size B, each sample: $\tau_i \sim D$, $s_1, s_2 \sim \tau_i$

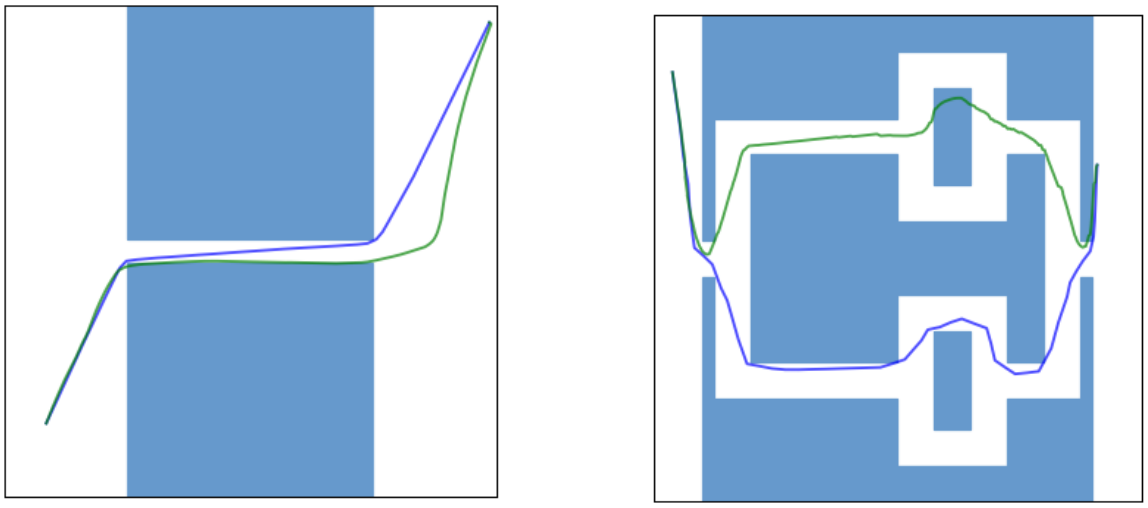
$s_m \leftarrow$ Get midpoint of $[s_1, s_2]$ according to τ_i for all items in batch

Update θ by minimizing the negative log-likelihood loss:

$$L = -\frac{1}{B} \cdot \sum_{b=1}^{B} \nabla_\theta \log P_{\hat{\pi}}(s_m^b|s_1^b, s_2^b; \theta)$$

end

return θ



(a) (b)

Figure 6. IL Experiment domains and results. The *simple* domain (a) and *hard* domain (b). A point robot should move only on the white free-space from a room on one side to the room on the other side while avoiding the blue obstacles. SGT plan (blue) executed successfully, *sequential* plan (green) collides with the obstacles.

	Success Rate		Prediction Times (Seconds)		Severity	
	Sequential	SGT	Sequential	SGT	Sequential	SGT
Simple	0.541	**0.946**	487.179	**28.641**	**0.0327**	0.0381
Hard - 2G	0.013	**0.266**	811.523	**52.22**	0.0803	**0.0666**
Hard - 4G	0.011	**0.247**	1052.62	**53.539**	0.0779	**0.0362**

Table 3. Results for IL experiments.

For both scenarios we collected a set of 111K (100K train + 10K validation + 1K test) expert trajectories from random start and goal states using a state-of-the-art motion planner (OMPL's(Şucan et al., 2012) Lazy Bi-directional KPIECE (Şucan & Kavraki, 2009) with one level of discretization). To account for different trajectory modalities, we chose a Mixture Density Network (MDN)(Bishop, 1994) as the parametric distribution of the predicted next state, both for the *sequential* and the SGT representations. We train the MDN by maximizing likelihood using Adam (Kingma & Ba, 2014). To ensure the same model capacity for both representations, we used the same network architecture, and both representations were trained and tested with the same data. Since the dynamical system is Markovian, the current and goal states are sufficient for predicting the next state in the plan, so we truncated the state history in the sequential model's input to contain only the current state.

For *simple*, the MDNs had a uni-modal multivariate-Gaussian distribution, while for *hard*, we experimented with 2 and 4 modal multivariate-Gaussian distributions, denoted as *hard-2G*, and *hard-4G*, respectively. As we later show in our results the SGT representation captures the demonstrated distribution well even in the *hard-2G* scenario, by modelling a bi-modal sub-goal distribution.

We evaluate the models using unseen start-goal pairs taken from the test set. To generate a trajectory, we predict sub-goals, and connect them using linear interpolation. We call a trajectory successful if it does not collide with an obstacle en-route to the goal. For a failed trajectory, we further measure the *severity* of collision by the percentage of the trajectory being in collision.

Table 3 summarizes the results for both representations. The SGT representation is superior in all three evaluation criteria - motion planning success rate, trajectory prediction times (total time in seconds for the 1K trajectory predictions), and severity. Upon closer look at the two *hard* scenarios, the *Sub Goal Tree* with a bi-modal MDN outperforms *sequential* with 4-modal MDN, suggesting that the SGT trajectory decomposition better accounts for multi-modal trajectories.

model	# Sub-goals	Success rate
SGT-PG	1	0.646±0.053
	3	0.663±0.153
	7	**0.696±0.116**
SeqSG	1	0.616±0.013
	3	0.576±0.013
	7	0.49±0.02

Table 4. Success rate on the NMP poles scenario

E. Technical Details for NMP Experiments Section 6.2

In this appendix we provide the full technical details of our policy gradient experiments presented in Section 6.2. We start by providing the explicit formulation of the NMP environment. We further provide a sequential-baseline not presented in the main text and its NMP environment and experimental results related to this baseline. We also present another challenging scenario – *poles*, and the results of *SGT-PG* and *SeqSG* in it. Finally, we conclude with modeling-related technical details [11].

E.1. Neural Motion Planning Formulation

In Section 6.2, we investigated the performance of *SGT-PG* and *SeqSG* executed in a NMP problem for the 7DoF Franka Panda robotic arm. That NMP formulation, denoted here as *with-reset*, allows for independent evaluations of motion segments, a requirement for FD-MSGT. The state $s \in \mathbb{R}^9$ describes the robot's 7 joint angles and 2 gripper positions, normalized to $[-1, 1]$. The models, *SGT-PG* and *SeqSG*, are tasked with predicting the states (sub-goals) in the plan.

A segment (s, s') is evaluated by running a PID controller tracking a linear joint motion from s to s'. Let s_{PID} denote the state after the PID controller was executed, and let α_{free} and $\alpha_{collision}$ be hyperparameters. We next define a cost function that encourages shorter paths, and discourages collisions; if a collision occurred during the PID execution, or 5000 simulation steps were not enough to reach s', a cost of $\alpha_{free} \cdot ||s - s_{PID}||_2 + \alpha_{collision} \cdot ||s' - s_{PID}||_2$ is returned. Otherwise, the motion was successful, and the segment cost is $\alpha_{free} \cdot ||s - s'||_2$. In our experiments we set $\alpha_{free} = 1$ and $\alpha_{collision} = 100$. Figures 9 and 10 show the training curves for the *self-collision* and *wall* scenarios respectively.

E.2. The *SeqAction* Baseline

We also evaluated a sequential baseline we denoted as *SeqAction*, which is more similar to classical RL approaches. Instead of predicting the next state, *SeqAction* predicts the next action to take (specifically the joint angles difference), and a position controller is executed for a single time-step instead of the PID controller of the previous methods.

SeqAction operates on a finite-horizon MDP (with horizon of $T = 5000$ steps). We follow the sequential NMP formulation of Jurgenson & Tamar (2019); the agent gets a high cost on collisions, high reward when reaching close to the goal, or a small cost otherwise (to encourage shorter paths). We denote this NMP variant *no-reset* as segments are evaluated sequentially, without resetting the state between evaluations. To make the problem easier, after executing an action in *no-reset*, the velocity of all the links is manually set to 0.

Similarly to Jurgenson & Tamar (2019), we found that sequential RL agents are hard to train for neural motion planning with varying start and goal positions without any prior knowledge or inductive bias. We trained *SeqAction* in the *self-collision* scenario using PPO (Schulman et al., 2017), and although we achieved 100% success rate when fixing both the start and the goal (to all zeros and all ones respectively), the model was only able to obtain less than 0.1 success rate when varying the goal state, and showed no signs of learning when varying both the start and the goal.

E.3. The *poles* Scenario

We tested both *SGT-PG* and *SeqSG* on another scenario we call *poles* where the goal is to navigate the robot between poles as shown in Figures 7 and 8. Similar to the previous scenarios, Table 4 shows the success rates of both models for 1, 3, and 7 sub-goals. Again, we notice that *SGT-PG* attains much better results and improves as more sub-goals are added, while the performance of *SeqSG* deteriorates. Moreover, even for a single sub-goal, *SGT-PG* obtains better success rates compared to

[11] The code is attached to the submission (a GitHub project will be posted here upon acceptance)

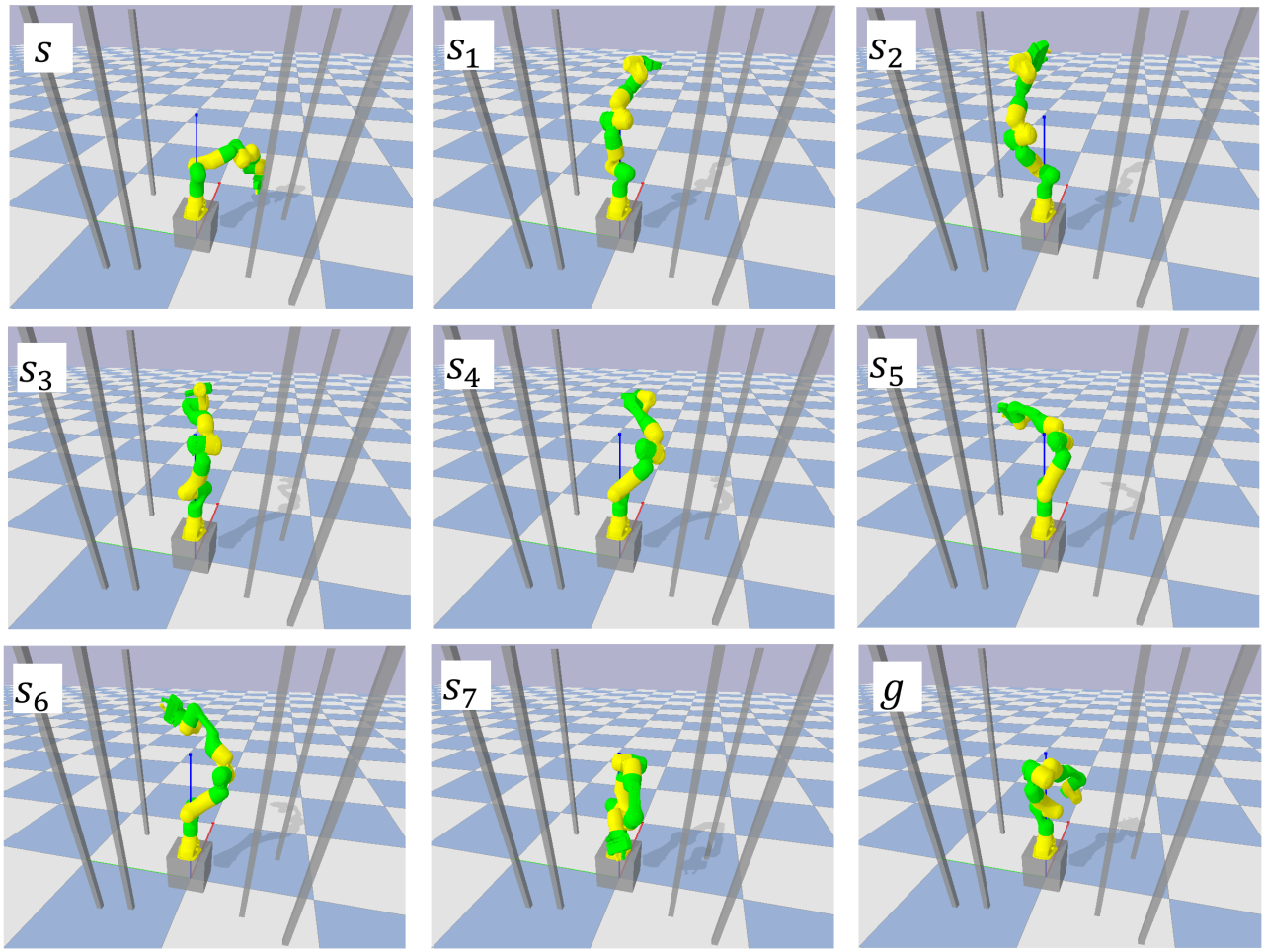


Figure 7. Robot trajectory *poles* scenario predicted with *SGT-PG* with 7 sub-goals. Top left start state, bottom right goal state. Notice that the two final segments s_6 to s_7 and s_7 to g are longer than the rest, but are made possible by the robot getting into position in state s_6. Note that *SGT-PG* failed to find a trajectory for this (s, g) pair with less sub-goals, indicating the hardness of the path.

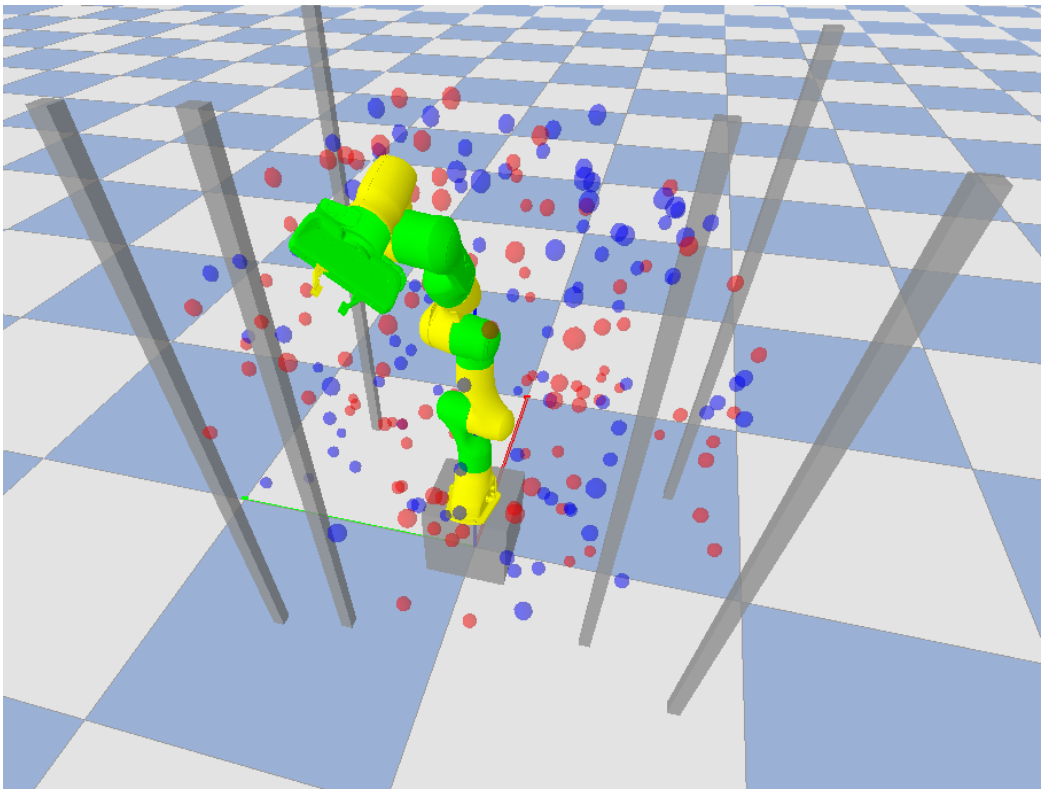


Figure 8. Start (blue) and goal (red) states in the test set of the *poles* scenario. Note that the agents are not exposed to these particular states during training.

SeqSG. The high range displayed by *SGT-PG* is due to one random seed that obtained significantly lower scores.

E.4. Neural Network Architecture and Training Procedure

We next provide full technical details regarding the architecture and training procedure of *SGT-PG* and *SeqSG* described in Section 6.2.

Architecture and training of *SGT-PG*: The start and goal states, s and g, are fed to a neural network with 3 hidden-layers of 20 neurons each and *tanh* activations. The network learns a multivariate Gaussian-distribution over the predicted sub-goal with a diagonal covariance matrix. Since the covariance matrix is diagonal, the size of the output layer is $18 = 2 \cdot 9$, 9 elements learn the bias, and 9 learn the standard deviation in each direction. To predict the bias, a value of $\frac{s+g}{2}$ is added to the prediction of the network. To obtain the final standard deviation, we add two additional elements to the standard deviation predicted by the neural network : (1) a small value (0.05), to prevent learning too narrow distributions, and (2) a learnable coefficient in each of the directions that depends on the distance between s and g, which allows large segments to predict a wider spread of sub-goals. These two changes were incorporated in order to mitigate log-likelihood evaluation issues.

SGT-PG trains on 30 (s, g) pairs per training cycle, using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.005 on the PPO objective (Schulman et al., 2017) augmented with a max-entropy loss term with coefficient 1, and a PPO trust region of $\epsilon = 0.2$. As mentioned in Section 5.2, when training level d for a single (s, g) pair, we sample M sub-goals (in our experiments $M = 10$) from the multivariate Gaussian described above, take the mean predictions of policies of depth $d - 1$ and lower, and take the mean of costs as a (s, g)-dependent baseline. The repetition allows us to obtain a stable baseline without incorporating another learnable module – a value function. We found this method to be more stable than training on 300 different start-goal pairs (no repetitions). During test time we always take the mean prediction generated by the policies.

Architecture of *SeqSG*: This model has two components, a policy and a value function. The policy architecture is identical to that of *SGT-PG*. Note that only a single policy is maintained as the optimal policy is independent of the remaining

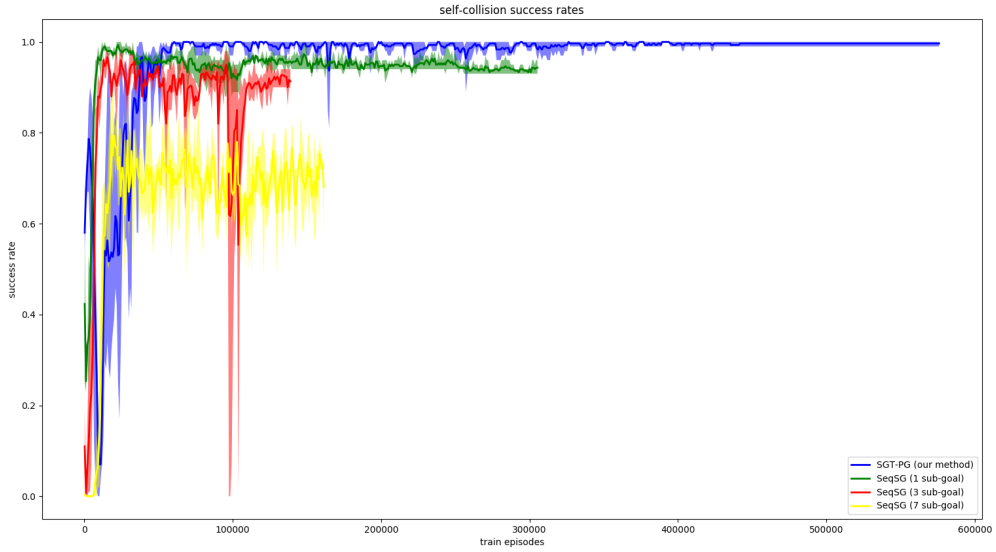


Figure 9. Success rates over training episodes of the *self-collision* scenario.

horizon[12]. We also found that learning the covaraince matrix causes stability issues for the algorithm, so the multivariate Gaussian only learns a bias term. The value function of *SeqSG* has 3 layers with 20 neurons each, and Elu activations. It predicts a single scalar which approximates $V(s, g)$, and is used as a state-dependent baseline to reduce the variance of the gradient estimation during training.

In order to compare apples-to-apples, *SeqSG* also trains on 30 (s, g) pairs with 10 repetitions each. We trained the policy with the PPO objective (Schulman et al., 2017) with trust region of $\epsilon = 0.05$. We used the Adam optimizer (Kingma & Ba, 2014), with learning rate of 0.005 but we clipped the gradient with L2 norm of 10 to stabilize training. The value function is trained to predict the value with mean-squared error loss. We also trained it with Adam with a learning rate of 0.005 but clipped the gradient L2 norm to 100.

We note that finding stable parameters for *SeqSG* was more challenging than for *SGT-PG*, as apparent by the gradient clippings, the lower trust region value, and our failure to learn the mutivariate Gaussian distribution covariance matrix.

[12] Also, a policy-per-time-step approach will not scale to larger plans, as we cannot maintain $T = 2^D$ different policies.

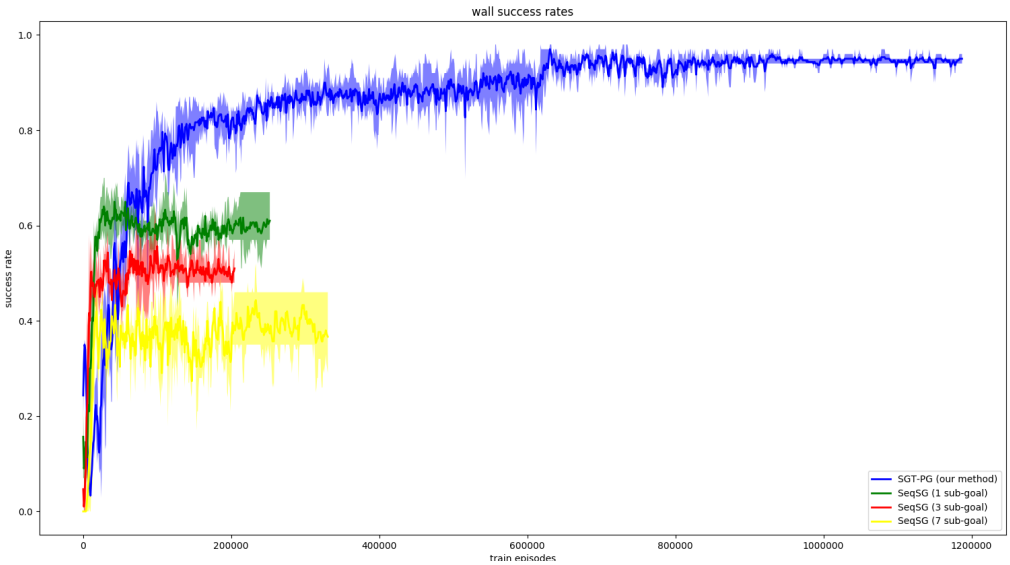


Figure 10. Success rates over training episodes of the *wall* scenario.