Planning in Hierarchical Reinforcement Learning: Guarantees for Using Local Policies

Tom Zahavy\textsuperscript{1,2}, Avinatan Hasidim\textsuperscript{1,3}, Haim Kaplan\textsuperscript{1,4}, Yishay Mansour\textsuperscript{1,4}

\textit{Google\textsuperscript{1}}, \textit{Technion Israel Institute of Technology\textsuperscript{2}, Bar Ilan University\textsuperscript{3}, Tel Aviv University\textsuperscript{4}}

\{TOMZAHAVY,AVINATAN,HAIMK,MANSOUR\} @GOOGLE.COM

Abstract

We provide theoretical guarantees for reward decomposition in deterministic MDPs with collectible rewards. Reward decomposition is a special case of hierarchical reinforcement learning, in which we view the reward as a sum, and we assemble a policy from policies for its components. Our approach builds on formulating this problem as a maximum traveling salesman problem with discounted reward. In particular, we focus on approximate solutions that are local, i.e., solutions that only observe information about the current state. Local policies are easy to implement and do not require substantial computational resources since they do not perform learning nor planning. Local deterministic policies, like Nearest Neighbor (NN), are being used in practice for hierarchical reinforcement learning, in particular, for 3D navigation. We propose three stochastic policies and prove that they guarantee better performance than any deterministic policy in the worst case. We then show experimentally that these policies outperform NN in deterministic MDPs with optimal options, and also in stochastic MDPs and during learning.

1. Introduction

One of the unique characteristics of human problem solving is the ability to represent the world on different granularities. When we plan a trip, we first choose the destinations we want to visit and only then decide what to do at each destination. Hierarchical reasoning enables us to map the complexities of the world around us into simple plans that are computationally tractable to reason. Nevertheless, the most successful Reinforcement Learning (RL) algorithms are still performing planning with only one abstraction level.

RL provides a general framework for optimizing decisions in dynamic environments. However, scaling it to real-world problems suffers from the curses of dimensionality; that is, coping with exponentially large state spaces, action spaces, and long horizons. One approach deals with large state spaces by introducing a function approximation to the value function or policy, making it possible to generalize across different states. Two famous examples are TD-Gammon (Tesauro, 1995) and the Deep Q Network (Mnih et al., 2015), both introduced a Deep Neural Network to approximate the value function leading to a high performance in solving Backgammon and video games. A different approach deals with long horizons by using a policy network to search among game outcomes efficiently (Silver et al. 2016), leading to a super-human performance in playing Go, Chess, and Poker (Silver et al. 2017; Moravčík et al. 2017). However, utilizing this approach when it is not possible to simulate the environment is still an open problem (Oh et al. 2015).

A long-standing approach for dealing with long horizons is to introduce hierarchy into the problem (Barto and Mahadevan, 2003). We will focus on the options framework (Sutton et al., 1999), a two-level hierarchy formulation, where options (local policies that map states to actions) are learned to achieve subgoals, while the policy over options selects among options to accomplish the
final goal of a task. Recently, it was demonstrated that learning a selection rule among pre-defined options using a DNN delivers promising results in challenging environments like Minecraft and Atari (Tessler et al., 2017; Kulkarni et al., 2016; Oh et al. 2017); other studies have shown that it is possible to learn options jointly with a policy-over-options end-to-end (Bacon et al., 2017).

In this work, we focus on a specific type of hierarchy - reward function decomposition - that dates back to the works of (Humphrys, 1996; Karlsson, 1997) and was recently combined with deep learning (van Seijen et al., 2017). In this formulation, each option \( i \) learns to maximize a local reward function \( R_i \), while the final goal is to maximize the sum of rewards \( R_M = \sum R_i \). Each option is trained separately, and provides a value function \( v_i \) and an option policy \( o_i \). The policy over options then uses these options and value functions to select among options. That way, each option is responsible for solving a simple task, and the options are learned in parallel across different machines. While the higher level policy can be trained using SMDP algorithms (Sutton et al., 1999), different research groups suggested using pre-defined rules to select among options. For example, choosing the option with maximal value function (Humphrys, 1996; Barreto et al., 2017), or choosing the action that maximizes the sum of option value functions (Karlsson, 1997). By using pre-defined rules, we can derive policies for MDP \( M \) by learning options (and without learning in MDP \( M \)), such that learning is fully decentralized. Although in many cases one can reconstruct the original MDP from the options, doing it would defeat the entire purpose of using options.

Even more specifically, we consider a set of \( n \) MDPs \( \{M_i\}_{i=1}^{n} \) with deterministic dynamics that share all components but the reward. Given a set of options, one per reward, with an optimal policy for collecting the reward, we are interested in deriving an optimal policy for collecting all the rewards, i.e., solving MDP \( M = \{S, A, P, R_M\} \) where \( R_M = \sum R_i \). In this setting, an optimal policy for \( M \) can be derived by solving the SMDP \( M_s = \{S, O, P, R_M\} \), whose actions are the optimal policies for collecting single rewards. We focus on collectible rewards, a special type of reward that is very common in 2D and 3D navigation domains like Minecraft (Tessler et al., 2017), DeepMindLab (Beattie et al., 2016) and VizDoom (Kempka et al., 2016). The challenge with collectible rewards is that the state changes each time we collect a reward (one can think of the subset of available rewards as part of the state). Since all the combinations of remaining items have to be considered, the state space grows exponentially with the number of rewards.

Here, we show that solving an SMDP under these considerations is equivalent to solving a Reward Discounted Traveling Salesman Problem (RD-TSP). Similar to the classical Traveling Salesman Problem (TSP), the goal in the RD-TSP is to find the best sequence to collect all the rewards (visit the cities); but instead of finding the shortest path, the goal is to maximize the discounted cumulative sum of rewards (Definition 3). Not surprisingly, computing an optimal solution to the RD-TSP is NP-hard (Blum et al. 2007).

A brute force approach for solving the RD-TSP requires evaluating all the \( n! \) possible tours connecting the \( n \) rewards. Instead, we adapted the BellmanHeldKarp dynamic programming algorithm for TSP (Bellman 1962, Held and Karp, 1962) to solve RD-TSP (see Algorithm 4 in the appendix). Surprisingly, this scheme is identical to tabular Q-learning on SMDP \( M_s \), and requires exponential time. This makes the task of computing the optimal policy for our SMDP infeasible.\(^1\) Blum et al. proposed a polynomial time planning algorithm for RD-TSP that computes a policy which collects at least 0.15 \( 0.15 \) fraction of the optimal discounted return, which was later improved to 0.19 (Farbstein and Levin 2016). These planning algorithms need to know the entire SMDP to

\(^1\) The Hardness results for RD-TSP do not rule out efficient solutions for special MDPs. E.g., we provide, in the appendix, exact polynomial-time solutions for the case in which the MDP is a line and when it is a star.
compute their approximately optimal policies. In contrast, in this work, we focus on deriving and analyzing policies that use only local information\(^2\) to make decisions; such local policies are more straightforward to implement and are more computationally efficient. The reinforcement learning community is already using simple local approximation algorithms for hierarchical RL. We hope that our research will provide crucial theoretical support for comparing local heuristics, and in addition introduce new reasonable local heuristics. Specifically, we prove worst-case guarantees on the reward collected by these algorithms relative to the optimal solution (given optimal options). We also prove bounds on the maximum reward that such local policies can collect. In our experiments, we compare the performance of these local policies in the planning setup (where all of our assumptions hold), and also during learning (when options are suboptimal), and in a stochastic environment.

**Our results:** We establish impossibility results for local policies, showing that no deterministic local policy can guarantee a reward larger than \(24\text{OPT}/n\) for any MDP, and no stochastic policy can guarantee a reward larger than \(8\text{OPT}/\sqrt{n}\) for every MDP (where \(\text{OPT}\) denotes the value of the optimal solution). These impossibility results imply that the Nearest Neighbor (NN) algorithm that iteratively collects the closest reward (and thereby a total of at least \(\text{OPT}/n\) reward) is optimal up to constant factor amongst all deterministic local policies.

On the positive side, we propose three simple stochastic policies that outperform NN. The best of them combines NN with a Random Depth First Search (RDFS) and guarantees performance of at least \(\Omega(\text{OPT}/\sqrt{n})\) when \(\text{OPT}\) achieves \(\Omega(n)\), and at least \(\Omega(\text{OPT}/n^{2/3})\) in the general case. Combining NN with jumping to a random reward and sorting the rewards by their distance from it, has a slightly worse guarantee. A simple modification of the NN to first jump to a random reward and continues NN from there, already improves the guarantee to \(O(\text{OPT}\log(n)/n)\).

2. Problem formulation

We now define our problem explicitly, starting from a general transfer framework in Definition 1, and then the more specific setting of collectible reward decomposition in Definition 2.

**Definition 1 (General Transfer Framework)** Given a set of MDPs \(\{M_i\}_{i=1}^n = \{S, A, P, \gamma, R_i\}_{i=1}^n\) and an MDP \(M = \{S, A, P, \gamma, R_M = f(R_1, ..., R_n)\}\) that differ only by their reward signal, derive an optimal policy for \(M\) given the optimal policies for \(\{M_i\}_{i=1}^n\).

Definition 1 describes a general transfer learning problem in RL. Similar to (Barreto et al., 2017), our framework assumes a set of MDPs sharing all but the reward signal. We are interested in transfer learning, i.e., using quantities that were learned from the MDPs \(\{M_i\}_{i=1}^n\) on the MDP \(M\). More specifically for model-free RL, given a set of optimal options and their value functions \(\{o_i^*, V_i^*\}_{i=1}^n\), we are interested in zero-shot transfer to MDP \(M\), i.e., deriving policies for solving \(M\) without learning in \(M\).

**Definition 2 (Collectible Reward Decomposition MDP)** An MDP \(M\) that satisfies the following properties: (1) **Reward Decomposition**, the reward in \(M\) represents the sum of the local rewards: \(R_M = \sum_{i=1}^n R_i\); (2) **Collectible Rewards**, each reward signal \(\{R_i\}_{i=1}^n\) represents a single collectible reward, i.e., \(R_i(s, a) = 1\) iff \(s = s_i\), \(a = a_i\) for some particular state \(s_i\) and action \(a_i\), and \(R_i(s, a) = 0\) otherwise. In addition, each reward can only be collected once; (3) **Deterministic Dynamics**, \(P\) is a deterministic transition matrix, i.e., for each action \(a\), each row of \(P^n\) has exactly one value that equals 1, and all other values equal zero.

---

2. Observe only the value of each option from the current state.
Property 1 in Definition 2 requires $R_M$ to be a decomposition of the previous rewards, and Property 2 requires each local reward to be a collectible reward. While limiting the generality, models that satisfy these properties have been vastly investigated (Oh et al., 2017; Barreto et al., 2017; Tessler et al., 2017; Higgins et al., 2017; van Seijen et al., 2017). Now, given that the value functions of the local policies are optimal, the shortest path from a reward $i$ to a reward $j$ is given by following option $o_j$ from reward-state $i$. In addition, the value function of the optimal policy of option $j$ at reward-state $i$ is $V_j(i) = \gamma^{d_{i,j}}$ where $d_{i,j}$ is the length of the shortest path from $i$ to $j$. Notice that at any state, an optimal policy on $M$ will always follow the shortest path to one of the rewards.  

Property 3 in Definition 2 requires deterministic dynamics. This property is perhaps the most limiting of the three, but again, it appears in numerous domains including many maze navigation problems, the Arcade Learning Environment (Bellemare et al., 2013), and games like Chess and Go. Given that $P$ is a deterministic transition matrix, an optimal policy on $M$ will make decisions only at states which contain rewards. In other words, once the policy arrived at a reward-state $i$ and decided to go to a reward-state $j$, it will follow the optimal policies $\pi_j$ until it reaches $j$.

For collectible reward decomposition (Definition 2), an optimal policy for $M$ can be derived on an SMDP (Sutton et al., 1999) denoted by $M_s$. The state space of $M_s$ contains only the initial state $s_0$ and the reward states $\{s_i\}_{i=1}^n$. The action space is replaced by the set of options $\{o_i\}_{i=1}^n$, where $o_i$ corresponds to following the optimal policy in $M_i$ until reaching state $s_i$. In addition, in this action space, the transition matrix $P$ is deterministic since $\forall s, a, s'$ such that $P_{s, a, s'} = 1$, and otherwise $P_{s, a, s'} = 0$. Finally, the reward signal and the discount factor remain the same.

We conclude this Section with Proposition 1, suggesting that an optimal policy on $M_s$ can be derived by solving an RD-TSP (Definition 3).

**Definition 3 (RD-TSP)** Given an undirected graph with $n$ nodes and edges of length $e_{i,j}$, find a path in the graph (defined as the set of indices $\{i_t\}_{t=1}^n$) that maximizes the discounted cumulative return: $\{i_t^*\}_{t=1}^n = \arg \max_{\{i_t\}_{t=1}^n \in \text{perm}\{1, \ldots, n\}} \sum_{j=0}^{n-1} \sum_{t=0}^j d_{e_{i_t,i_{t+1}}},$

**Proposition 1 (MDP to RD-TSP)** Given an MDP $M$ that satisfies Definition 2 with $n$ rewards and a set of options $\{o_i\}_{i=1}^n$ for collecting them, define a graph, $G$, with nodes corresponding to the initial state and the reward-states of $M$. Define the length $d_{e_{i,j}}$ of an edge $e_{i,j}$ in $G$ to be $V_j(i)$, i.e., the value of following option $o_j$ from state $i$. Then, an optimal policy in $M$ can be derived by solving an RD-TSP in $G$.

To summarize, our modeling approach allows one to deal with the curses of dimensionality in three different ways. First, each option can be learned with function approximation techniques, e.g., (Tessler et al., 2017; Bacon et al., 2017), to deal with raw, high dimensional inputs like vision and text. Second, formulating the problem as an SMDP reduces the state space to include only the reward states and effectively reduces the planning horizon (Sutton et al., 1999; Mann et al., 2015). Third, under the RD-TSP formulation, we derive approximate solutions to deal with the exponentially large state spaces that emerge from modeling one-time events like collectible rewards.

---

3. To see this, assume there exists a policy $\mu$ that does not follow the shortest path from some state $k$ to the next reward-state $k'$. Then, we can improve $\mu$ by taking the shortest path from $k$ to $k'$, contradicting the optimality of $\mu$. This implies that an optimal policy on $M$ is a composition of the local options $\{o_i\}_{i=1}^n$.

4. Note that this is not true if $P$ is stochastic. To see this, recall that stochastic shortest path is only shortest in expectation. Thus, stochasticity may lead to states that require changing the decision.
3. Local heuristics

We consider the SMDP $M_s$ described by the complete graph $G$ derived from $M$ in Proposition 1. Recall that we have a reward in each vertex of $G$. We start by defining a local policy. All the policies which we analyze are local policies. A local policy is a mapping who inputs are: (1) the current state $x$, (2) the history $h$ containing the previous steps taken by the policy; in particular $h$ includes the rewards that we have already collected, and (3) the discounted return for each option from the current state, i.e., $\{V_i(x)\}_{i=1}^n$, and whose output is a distribution over the options. Notice that a local policy does not have full information on the MDP (but only on local distances).

**Definition 4 (Local policy)** A local policy $\pi_{local}$ is a mapping:

$$\pi_{local}(x, h, \{V_i(x)\}_{i=1}^n) \rightarrow \Delta(\{o_i\}_{i=1}^n),$$

where $\Delta(X)$ is the set of distributions over a finite set $X$.

3.1 NN performance

We start with an analysis of one of the most natural heuristics for the TSP, the famous NN algorithm. In the context of our problem, NN is the policy that selects the option with the highest estimated value, exactly like GPI (Barreto et al., 2017). We shall abuse the notation slightly and use the same name (e.g., NN) for the algorithm itself and its value; no confusion will arise. For TSP (without discount) in general graphs, we know that

$$\frac{1}{3} \log_2(n+1) + \frac{4}{3} \leq \frac{NN}{OPT} \leq \frac{\log_2(n)}{2} + \frac{1}{2}$$

(Rosenkrantz et al., 1977). However, for RD-TSP the NN algorithm only guarantees a value of $\frac{OPT}{n}$, as Theorem 5 states. In the next subsection, we prove a lower bound for deterministic local policies (such as NN) of $O(\frac{OPT}{n})$. This implies that NN is optimal for deterministic policies.

**Theorem 5 (NN Performance)** For any MDP satisfying Definition 2 with $n$ rewards, and any discount factor $\gamma$:

$$\frac{NN}{OPT} \geq \frac{1}{n}.$$ 

Next, we propose a simple, easy to implement, stochastic adjustment to the vanilla NN algorithm with a better upper bound which we call R-NN (Random-NN). The algorithm starts by collecting one of the rewards, say $s_1$, at random, and continues by executing NN (Algorithm 1). The following theorem shows that our stochastic modification to NN improves its guarantees by a factor of $\log(n)$.

While the improvement over NN may seem small ($\log(n)$) the observation that stochasticity improves the performance guarantees of local policies is essential to our work. In the following sections, we derive more sophisticated randomized algorithms with better performance guarantees.

**Theorem 6 (R-NN Performance)** For an MDP satisfying Definition 2 with $n$ rewards:

$$\frac{R\text{-NN}}{OPT} \geq \Omega\left(\frac{\log(n)}{n}\right).$$

**Algorithm 1** R-NN: NN with a First Random Pick

- **Input**: MDP $M$, with $n$ rewards, and $s_0$ the first reward
- Flip a coin
- if outcome = heads then {Perform NN}
  - Collect a random reward, denote it by $s_1$
- end if
- Follow by executing NN

5. Notice that the optimal global policy, (computed given the entire SMDP as input), has the same history dependence as local policies. However, while the global policy is deterministic, our results show that for local policies, stochastic policies are better than deterministic ones. It follows that the benefit of stochastic selection is due to the local information and not due to the dependence on the history. This conclusion implies that locality may be related to partial observability, for which it is known that stochastic policies can be better (Aumann et al., 1996).
3.2 Impossibility Results

In the previous section, we saw that the NN heuristic guarantees performance of at least \( \frac{\text{OPT}}{n} \). Next, we show an impossibility result for all deterministic local policies, indicating that no such policy can guarantee more than \( \frac{\text{OPT}}{n} \), which makes NN optimal over such policies.

**Theorem 7 (Impossibility for Deterministic Local Policies)** For any deterministic local policy \( D-\text{Local} \), there exists an MDP with \( n \) rewards and a discount factor \( \gamma = 1 - \frac{1}{n} \) such that: \( \frac{D-\text{Local}}{\text{OPT}} \leq \frac{24}{n} \).

This Theorem implies that NN is optimal over local deterministic policies, and that a small stochastic adjustment can improve its guarantees. These observations motivated us to look for better local policies in the broader class of stochastic local policies. Theorem 8 provides better impossibility result for such policies.

**Theorem 8 (Impossibility for Stochastic Local Policies)** For each stochastic local policy \( S-\text{Local} \), there exists an MDP with \( n \) rewards and a discount factor \( \gamma = 1 - \frac{1}{\sqrt{n}} \) such that: \( \frac{S-\text{Local}}{\text{OPT}} \leq \frac{8}{\sqrt{n}} \).

We do not have a policy that achieves this lower bound, but we now propose and analyze two stochastic policies (in addition to the R-NN) that substantially improve over the deterministic upper bound. As we will see, these policies satisfy the Occam’s razor principle, i.e., policies with better guarantees are also more complicated and require more computational resources.

3.3 NN with Randomized DFS (RDFS)

We now describe the NN-RDFS policy (Algorithm 2), our best performing local policy. The policy performs NN with probability 0.5 and local policy which we call RDFS with probability 0.5. RDFS first collects a random reward and continues by performing a DFS on edges shorter than \( \theta \), where \( \theta \) is chosen at random. When it runs out of edges shorter than \( \theta \) then RDFS continues by performing NN. The performance guarantees for the NN-RDFS method are stated in Theorem 9.

**Theorem 9 (NN-RDFS Performance)** For any MDP that satisfies Definition 2 with \( n \) rewards, \( \frac{\text{NN-RDFS}}{\text{OPT}} \geq \begin{cases} \Omega\left(\frac{n^{\frac{1}{2}}}{\log^2(n)}\right), & \text{if OPT = } \Omega(n) \ , \\ \Omega\left(\frac{n^{\frac{3}{4}}}{\log^2(n)}\right), & \text{otherwise} \ . \end{cases} \)

**Proof sketch.** The analysis is conducted in three steps. In the first two steps, we assume that OPT achieved a value of \( \Omega(n') \) by collecting \( n' \) rewards at a segment of length \( x \leq \log^2(2) \).

The first step considers the case where \( n' = \Omega(n) \), and in the second step we remove this requirement and analyze the performance of NN-RDFS for the worst value of \( n' \). The third step considers all the rewards collected by OPT (not necessarily in a segment of length \( x \)) and completes the proof. In the second and the third steps we loose two logarithmic factors. One since we use a segment of length \( x \) in which OPT collects value of at least \( \text{OPT}/\log(n) \), and the second for guessing a good enough approximation for \( n' \) (for setting \( \theta \)).
3.4 NN with a Random Ascent (RA)

We now describe the NN-RA policy (Algorithm 3). Similar in spirit to NN-RDFS, the policy performs NN with probability 0.5 and local policy which we call RA with probability 0.5. RA starts at a random node, \( s_1 \), sorts the rewards in increasing order of their distance from \( s_1 \) and then collects all other rewards in this order. The algorithm is simple to implement, as it does not require guessing any parameters (like \( \theta \) which RDFS has to guess). However, this comes at the cost of a worse bound.

**Theorem 10 (NN-RA Performance)**

For any MDP that satisfies Definition 2 with \( n \) rewards and for any discount factor \( \gamma \):

\[
\frac{\text{NN-RA}}{\text{OPT}} \geq \begin{cases} 
\Omega\left(\frac{n^{\frac{2}{3}}}{\log(n)}\right), & \text{if } \text{OPT} = \Omega(n). \\
\Omega\left(\frac{n^{\frac{4}{3}}}{\log(n)}\right), & \text{otherwise.}
\end{cases}
\]

The performance guarantees for the NN-RA method are given in Theorem 10. The analysis follows the same steps as the proof of the NN-RDFS algorithm. We emphasize that here, the pruning parameter \( \theta \) is only used for analysis purposes and is not part of the algorithm. Consequently, we see only one logarithmic factor in the performance bound of Theorem 10 in contrast with two in Theorem 9.

4. Simulations

We evaluated our algorithms in an MDP that satisfies Definition 2, and in generalized more challenging settings in which the MDP is stochastic. We also evaluated them throughout the learning process of the options, when the option-policies (and value functions) are sub-optimal. Note that with sub-optimal options the agent may reach the reward in sub-optimal time and even may not reach the reward at all. Furthermore, in stochastic MDPs, while executing an option the agent may arrive to a state where it prefers to switch to a different option rather than completing the execution of the current option. Our experiments show that our policies perform well, even when some of our assumptions for our theoretical analysis are relaxed. We note that a second suite of (planning) experiments is included in the supplementary material, where we evaluate the local policies in the setup of Definition 2 on different MDPs. In these simulations, we further distinguish between our policies and demonstrate settings in which it is better to use each policy.

**Setup.** An agent (yellow) is placed in a 50X50 grid-world domain (Figure, top left). Its goal is to navigate in the maze and collect the available 45 rewards (teal) as fast as possible. The agent can move by going up, down, left and right, and without crossing walls (red). In the stochastic scenario, there are also four actions, up, down, left and right, but once an action, say up, is chosen there is a 10% chance that a random action (chosen uniformly) will be executed instead of up. We are interested in testing our algorithms in the regime where \( \text{OPT} \) can collect almost all of the rewards within a constant discount (i.e., \( \text{OPT} \approx \alpha n \)), but, there also exist bad tours that achieve a constant value (i.e., taking the most distance reward in each step); thus, we set \( \gamma = 1 - \frac{1}{n} \). The agent consists of a set of options, and a policy over the options. We have an option per reward that learns, by interacting with the environment, a policy that moves the agent from its current position to the reward in the shortest distance.

We leave this theoretical analysis to future work.
way. We learned the options in parallel using Q-learning. We performed the learning in epochs. At each epoch, we initialized each option in a random state, and performed Q-learning steps either until the option found the reward or until $T = 150$ steps have passed. Every phase of $K = 2000$ epochs ($300k$ steps), we tested our policies with the available set of options. We performed this evaluation for $L = 150$ phases, resulting in a total of $45M$ training steps for each option. At the end of these epochs the policy for each option was approximately optimal.

**Options:** The Figure (top, right), shows the quality of the options in collecting the reward during learning in the stochastic MDP. For each of the $L$ phases of $K$ epochs ($300k$ steps) we plot the fraction of the runs in which the option reached the reward (red), and the option time gap (blue). The option time gap is the time that took the option to reach the goal, minus the deterministic shortest path from the initial state to the reward state. We can see that the options improve as learning proceeds, succeeding to reach the reward in more than 99% of the runs. The time gap converges, but not zero, since the shortest stochastic path (due to the 10% random environment and the $\epsilon$-greedy policy with $\epsilon = 0.1$) is longer than the shortest path. Similar results for a deterministic MDP can be found in the supplementary material, showing that the gap is 50% shorter (only the policy is stochastic).

**Local policies:** At the bottom of the Figure, we show the performance of the different policies (using the options available at the end of each of the $L$ phases), measured by the discounted cumulative return. In addition to our four policies, we also evaluated two additional heuristics for comparison. The first, denoted by RAND, is a random policy over the options, which selects an option at random at each step. Rand performs the worst since it goes in and out from clusters. The second, denoted by OPT (with a slight abuse of notation), is a fast approximation to OPT that uses all the information about the MDP (not local) and computes an approximation to OPT by checking all possible choices of the first two clusters and all possible paths through the rewards that they contain and picks the best. This is a good approximation for OPT since discounting makes the rewards collected after the first two clusters negligible. OPT performs better than our policies, because it has the knowledge on the full SMDP. On the other hand, our policies perform competitively, without learning the policy over options at all (zero shot solution). Among the local policies, we can see that NN is not performing well since it is "tempted" to collect nearby rewards in small clusters instead of going to large clusters. The stochastic algorithms, on the other hand, choose the first reward at random, thus they have a higher chance to reach larger clusters, and consequently, they perform better than NN. R-NN and NN-RDFS perform the best and almost the same, because effectively, inside the first two clusters RDFS is taking a tour which is similar to the one taken by NN. This happens because $\theta$ is larger than most pairwise distances inside clusters. NN-RA performs worse than the other stochastic algorithms, since sorting the rewards by their distances from the first reward in the cluster introduces an undesired "zig-zag" behavior, in which we do not collect rewards which are at approximately the same distance from the first in the right order (see the experiments in the supplementary material for visualization and more details).
References


Nathan Sprague and Dana Ballard. Multiple-goal reinforcement learning with modular sarsa (0). 2003.


5. NN performance

We note that by Proposition 1, solving an MDP that satisfies Definition 2, is equivalent to solving an RD-TSP. To make the proofs clearer, we use the RD-TSP notation, i.e., a graph $G$ corresponds to MDP $M$; nodes correspond to states; edges to value functions.

**Theorem 4 (NN Performance)** For any MDP satisfying Definition 2 with $n$ rewards, and any discount factor $\gamma$:

$$\frac{\text{NN}}{\text{OPT}} \geq \frac{1}{n}.$$ 

**Proof** Denote by $i^*$ the nearest reward to the origin $s_0$, and by $d_{0,i^*}$ the distance from the origin to $i^*$. The distance from $s_0$ to the first reward collected by OPT is at least $d_{0,i^*}$. Thus, if $o_0 = s_0, o_1, \ldots, o_{n-1}$ are the rewards ordered in the order by which OPT collects them we get that

$$\text{OPT} = \sum_{j=0}^{n-1} \gamma^{\sum_{t=0}^j d_{o_t,o_{t+1}}} \leq \gamma^{d_{0,i^*}} \left( 1 + \sum_{j=1}^{n-1} \gamma^{\sum_{t=1}^j d_{o_t,o_{t+1}}} \right) \leq n \gamma^{d_{0,i^*}}.$$ 

On the other hand, the NN heuristic chooses $i^*$ in the first round, thus, its cumulative reward is at least $\gamma^{d_{0,i^*}}$ and we get that

$$\frac{\text{NN}}{\text{OPT}} \geq \frac{\gamma^{d_{0,i^*}}}{n \gamma^{d_{0,i^*}}} = \frac{1}{n}.$$ 

6. Impossibility results

6.1 Deterministic Local Policies

**Theorem 6 (Impossibility Results for Deterministic Local Policies)** For any deterministic local policy $D$-Local, there exists an MDP with $n$ rewards and a discount factor $\gamma = 1 - \frac{1}{n}$ such that:

$$\frac{D$-Local}{\text{OPT}} \leq \frac{24}{n}.$$ 

**Proof** Consider a family of graphs, $G$, each of which consists of a star with a central vertex and $n$ leaves. The starting vertex is the central vertex, and there is a reward at each leaf. The length of each edge is $d$, where $d$ is chosen s.t. $\gamma^d = \frac{1}{2}$.

Each graph of the family $G$ corresponds to a different subset of $n/2$ of the leaves which we connect (pairwise) by edges of length 1. (The other $n/2$ leaves are only connected to the central vertex.) While at the central vertex, local policy cannot distinguish among the $n$ rewards (they all at the same distance from the origin), and therefore its choice is the same for all graphs in $G$. (The following choice is also the same and so on, as long as it does not hit one of the $n/2$ special rewards.)
It follows that, for any given policy, there exists a graph in $G$ such that the adjacent $n/2$ rewards are visited last. Finally, since $\gamma = 1 - \frac{1}{\sqrt{n}}$ we have that $\frac{n}{4} \leq \sum_{i=0}^{\frac{n}{2}-1} \gamma^i = \frac{1 - \gamma^{n/2}}{1 - \gamma} \leq \frac{n}{2}$ and thus

$$\frac{D-\text{Local}}{\text{OPT}} = \frac{\sum_{i=1}^{\frac{n}{2}} \gamma^{(2i-1)d} + \gamma^{nd+1} \sum_{i=0}^{\frac{n}{2}-1} \gamma^i}{\gamma d \sum_{i=0}^{\frac{n}{2}-1} \gamma^i + \gamma^{2d+\frac{n}{2}-1} \sum_{i=1}^{\frac{n}{2}} \gamma^{(2i-1)d}} \leq \frac{\sum_{i=1}^{\frac{n}{2}} \gamma^{(2i-1)d} + 0.5n \gamma^{nd+1}}{\gamma d \sum_{i=0}^{\frac{n}{2}-1} \gamma^i} = \frac{2 \sum_{i=1}^{\frac{n}{2}} 0.25^i + 0.5 \frac{n}{2}}{0.5 \sum_{i=0}^{\frac{n}{2}-1} \gamma^i} \leq \frac{6}{\sum_{i=0}^{\frac{n}{2}-1} \gamma^i} \leq \frac{24}{n}.$$ 


### 6.2 Stochastic Local Policies

**Theorem 7 (Impossibility Results for Stochastic Local Policies)** For each stochastic local policy $S$-Local, there exists an MDP with $n$ rewards and a discount factor $\gamma = 1 - \frac{1}{\sqrt{n}}$ such that:

$$\frac{S-\text{Local}}{\text{OPT}} \leq \frac{8}{\sqrt{n}}.$$

**Proof** We consider a family of graphs, $G$, each of which consists of a star with a central vertex and $n$ leaves. The starting vertex is the central vertex, and there is a reward at each leaf. The length of each edge is $d$, where $d$ is chosen such that $\gamma^d = \frac{1}{2}$. Each graph in $G$ corresponds to a subset of $\sqrt{n}$ leaves which we pairwise connect to form a clique.

Since $\gamma = 1 - \frac{1}{\sqrt{n}}$, we have that $\sum_{i=0}^{\sqrt{n}-1} \gamma^i \geq \frac{\sqrt{n}}{2}$, and therefore

$$\text{OPT} = \gamma^d \sum_{i=0}^{\sqrt{n}-1} \gamma^i + \gamma^{2d+\sqrt{n}-1} \sum_{i=1}^{n-\sqrt{n}} \gamma^{(2i-1)d} \geq 0.5 \sum_{i=0}^{\sqrt{n}-1} \gamma^i \geq 0.25\sqrt{n}.$$

On the other hand, local policy at the central vertex cannot distinguish among the rewards and therefore for every graph in $G$ it picks the first reward from the same distribution. The policy continues to choose rewards from the same distribution until it hits the first reward from the $\sqrt{n}$-size clique.

To argue formally that every $S$-Local policy has small expected reward on a graph from $G$, we use Yao’s principle (Yao, 1977) and consider the expected reward of a D-Local policy on the uniform distribution over $G$.

Let $p_1 = \sqrt{n}/n$ be the probability that D-Local picks its first vertex from the $\sqrt{n}$-size clique. Assuming that the first vertex is not in the clique, let $p_2 = \sqrt{n}/(n-1)$ be the probability that the second vertex is from the clique, and let $p_3, p_4, \ldots$ be defined similarly. When D-local picks a
vertex in the clique then its reward (without the cumulative discount) is \(O(\sqrt{n})\). However, each
time D-Local misses the clique then it collects a single reward but suffers a discount of \(\gamma^2 = 1/4\).
Neglecting the rewards collected until it hits the clique, the total value of D-Local is
\[
O \left( \left( p_1 + (1 - p_1) \gamma^{2d} p_2 + (1 - p_1)(1 - p_2) \gamma^{3d} p_3 \ldots \right) \sqrt{n} \right)
\]
Since \(p_i \leq 2/\sqrt{n}\) for \(1 \leq i \leq n/2\) this value is \(O(1)\). 

7. NN-RDFS

**Theorem 8 (NN-RDFS Performance)** For any MDP that satisfies Definition 2 with \(n\) rewards,

\[
\frac{\text{NN-RDFS}}{\text{OPT}} \geq \begin{cases} 
\Omega \left( \frac{n^{-\frac{1}{2}}}{\log^*(n)} \right), & \text{if OPT} = \Omega(n). \\
\Omega \left( \frac{n^{-\frac{2}{3}}}{\log^*(n)} \right), & \text{otherwise.}
\end{cases}
\]

**Proof** Step 1. Assume that OPT collects a set \(S_{\text{OPT}}\) of \(\alpha n\) rewards for some \(0 \leq \alpha \leq 1\), in a segment \(p\) of length \(x = \log_{\frac{1}{2}}(2)\) (i.e. \(x\) is the distance from the first reward to the last reward – it does not include the distance from the starting point to the first reward). Let \(d_{\text{min}}, d_{\text{max}}\) the shortest and longest
distances from \(s_0\) to a reward in \(S_{\text{OPT}}\) respectively. By the triangle inequality, \(d_{\text{max}} - d_{\text{min}} \leq x\). We further assume that \(OPT \leq O(\gamma^{d_{\text{min}} n})\) (i.e., That is the value that OPT collects from rewards which are not in \(S_{\text{OPT}}\) is negligible). We now show that RDFS is \(\Omega(\sqrt{n})\) for \(\theta = x/\sqrt{n}\). We start with the following Lemma.

**Lemma 11** For any path \(p\) of length \(x\), and \(\forall \theta \in [0, x]\), there are less than \(\frac{x}{\theta}\) edges in \(p\) that are
larger than \(\theta\).

**Proof.** For contradiction, assume there are more than \(\frac{x}{\theta}\) edges longer than \(\theta\). The length of \(p\) is given
by \(\sum_i p_i = \sum_{p_i \leq \theta} p_i + \sum_{p_i > \theta} p_i \geq \sum_{p_i \leq \theta} p_i + \frac{x}{\theta} \theta > x\) thus a contradiction to the assumption that
the path length is at most \(x\). 

Lemma 11 assures that after pruning all edges larger than \(\theta\) (from the graph), there are at most \(\frac{x}{\theta}\)
Connected Components (CCs) \(\{C_j\}_{j=1}^\frac{x}{\theta}\) in \(S_{\text{OPT}}\). In addition, it holds that \(\sum_{j=1}^\frac{x}{\theta} |C_j| = \alpha n\), and all
the edges inside any connected component \(C_j\) are shorter than \(\theta\).

Next, we (lower) bound the total gain of RDFS. Say that RDFS starts at a reward in component \(C_j\). Then, since all edges in \(C_j\) are shorter than \(\theta\), it collects either all the rewards in \(C_j\), or at least \(x/2\theta\) rewards. Thus, RDFS collects \(\Omega \left( \min \{|C_j|, \frac{x}{\theta} \} \right)\) rewards. To see this, recall that the DFS algorithm traverses each edge at most twice. In addition, as long as it did not collect all vertices of
\(C_j\), the length of each edge that the DFS traverses is at most \(\theta\). Therefore if the algorithm did not
collect all the vertices in this component in its prefix of length \(x\), then it collected at least \(x/(2\theta)\) rewards in this prefix. This gives the lower bound of \(\min \left( |C_j|, x/(2\theta) \right)\) on the number of rewards
that the algorithm collects in a prefix of length \(x\).
Notice that the first random step leads RDFS to a vertex in CC $C_j$ with probability $\frac{|C_j|}{n}$. If more than half of rewards are in CCs s.t $|C_j| \geq \frac{x}{\theta}$, then
\[
RDFS \geq \gamma_d^{\text{max}} \sum_{j=1}^{\frac{s}{\theta}} \frac{|C_j|}{n} \cdot \min\{\frac{|C_j|}{\theta}, \frac{x}{\theta}\} \geq \gamma_d^{\text{max}} \sum_{j:|C_j| \geq \frac{x}{\theta}} \frac{|C_j|}{n} \cdot \frac{x}{\theta} \geq \gamma_d^{\text{max}} \frac{\alpha x}{2\theta}.
\]

If more than half of rewards in $S_{\text{OPT}}$ are in CCs such that $|C_j| \leq \frac{x}{\theta}$, let $s$ be the number of such CCs and notice that $s \leq \frac{x}{\theta}$. We get that:
\[
RDFS = \gamma_d^{\text{max}} \sum_{j=1}^{s} \frac{|C_j|}{n} \cdot \min\{\frac{|C_j|}{\theta}, \frac{x}{\theta}\} \geq \gamma_d^{\text{max}} \sum_{j:|C_j| \leq \frac{x}{\theta}} \frac{|C_j|^2}{n} \geq \frac{s}{n} \gamma_d^{\text{max}} \left(\frac{1}{s} \sum_{j=1}^{s} |C_j|^2\right) \geq \gamma_d^{\text{max}} \frac{\theta \alpha^2 n}{4x}.
\]

By setting $\theta = \frac{x}{\sqrt{n}}$ we guarantee that the value of RDFS is at least $\gamma_d^{\text{max}} \alpha^2 \sqrt{n}/4$. Since $d_{\text{max}} - d_{\text{min}} \leq x$,
\[
\frac{RDFS}{\text{OPT}} \geq \gamma_d^{\text{max}} \frac{\alpha^2 \sqrt{n}/4}{\gamma_d^{\text{min}} \alpha n} \geq \frac{\alpha x}{4\sqrt{n}} = \frac{\alpha}{2\sqrt{n}},
\]
where the last inequality follows from the triangle inequality.

**Step 2.** Assume that OPT gets its value from $n' < n$ rewards that it collects in a segment of length $x$ (and from all other rewards OPT collects a negligible value). Recall that the NN-RDFS policy is either NN with probability 0.5 or RDFS with probability 0.5. By picking the single reward closest to the starting point, NN gets at least $1/n'$ of the value of OPT. Otherwise, with probability $n'/n$, RDFS starts with one of the $n'$ rewards picked by OPT and then, by the analysis of step 1, if it sets $\theta = \frac{x}{\sqrt{n'}}$, RDFS collects $\frac{1}{2\sqrt{n'}}$ of the value collected by OPT (we use Step (1) with $\alpha = 1$). It follows that
\[
\frac{\text{NN-RDFS}}{\text{OPT}} \geq \frac{1}{2} \cdot \frac{1}{n'} + \frac{1}{2} \cdot \frac{n'}{n} \cdot \frac{1}{2\sqrt{n'}} = \frac{1}{2n'} + \frac{\sqrt{n'}}{4n}.
\]

This lower bound is smallest when $n' \approx n^{2}$, in which case NN-RDFS collect $\Omega(n^{-2/3})$ of OPT.

First, notice that since $n'$ is not known to NN-RDFS, it has to be guessed in order to choose $\theta$. This is done by setting $n'$ at random from $n' = n/2^i, i \sim \text{Uniform}\{1, 2, ..., \log_2(n)\}$. This guarantees that with probability $\frac{1}{\log(n)}$ our guess for $n'$ will be off of its true value by a factor of at most 2. With this guess we will work with an approximation of $\theta$ which is off of its true value by a factor of at most $\sqrt{2}$. These approximations degrade our bounds by a factor of $\log(n)$.

**Step 3.** Finally, we consider the general case where OPT may collect its value in a segment of length larger than $x$. Notice that the value which OPT collects from rewards that follow the first $\log_2(n)$ segments of length $x$ in its tour is at most 1 (since $\gamma^{\log_2(n) \cdot x} = \frac{1}{x}$). This means that there exists at least one segment of length $x$ in which OPT collects at least $\frac{\text{OPT}}{\log_2(n)}$ of its value. Combining this with the analysis in the previous step, the proof is complete.  

\[\blacksquare\]
8. NN-RA

**Theorem 9 (NN-RA Performance)** For any MDP that satisfies Definition 2 with \( n \) rewards and for any discount factor \( \gamma \):

\[
\frac{\text{NN-RA}}{\text{OPT}} \geq \begin{cases} 
\Omega \left( \frac{n^{\frac{2}{3}}}{\log(n)} \right), & \text{if } \text{OPT} = \Omega(n). \\
\Omega \left( \frac{n^{\frac{2}{3}}}{\log(n)} \right), & \text{otherwise.}
\end{cases}
\]

**Proof Step 1.** Assume that OPT collects a set \( S_{\text{OPT}} \) of \( \alpha n \) rewards for some \( 0 \leq \alpha \leq 1 \), in a segment \( p \) of length \( x = \log_\frac{1}{\gamma}(2) \) (i.e. \( x \) is the distance from the first reward to the last reward – it does not include the distance from the starting point to the first reward). Let \( d_{\text{min}}, d_{\text{max}} \) the shortest and longest distances from \( s_0 \) to a reward in \( S_{\text{OPT}} \) respectively. By the triangle inequality, \( d_{\text{max}} - d_{\text{min}} \leq x \).

We further assume that \( \text{OPT} \leq O(\gamma d_{\text{min}} \alpha n) \) (i.e., That is the value that OPT collects from rewards which are not in \( S_{\text{OPT}} \) is negligible).

Let \( \theta \) be a threshold that we will fix below, and denote by \( \{C_j\} \) the CCs of \( S_{\text{OPT}} \) that are created by deleting edges longer than \( \theta \) among vertices of \( S_{\text{OPT}} \). By Lemma 11, we have at most \( \frac{x}{\theta} \) CCs.

Assume that RA starts at a vertex of a component \( C_j \), such that \( |C_j| = k \). Since the diameter of \( C_j \) is at most \( (|C_j| - 1)\theta \) then it collects its first \( k \) vertices (including \( s_1 \)) within a total distance of \( 2 \sum_{i=1}^{k} (i-1)\theta \leq k^2 \theta \). So if \( k^2 \theta \leq x \) then it collects at least \( |C_j| \) rewards before traveling a total distance of \( x \), and if \( k^2 \theta > x \) it collects at least \( \lfloor \sqrt{x/\theta} \rfloor \) rewards. (We shall omit the floor function for brevity in the sequel.) It follows that RA collects \( \Omega(\min\{|C_j|, \sqrt{x/\theta}\}) \) rewards. Notice that the first random step leads RDFS to a vertex in CC \( C_j \) with probability \( \frac{|C_j|}{n} \).

If more than half of rewards are in CCs \( s \) of rewards in \( S_{\text{OPT}} \) are in CCs such that \( |C_j| \leq \sqrt{x/\theta} \), let \( s \) be the number of such CCs and notice that \( s \leq \frac{x}{\theta} \). We get that:

\[
\text{RA} \geq \gamma^{d_{\text{max}}} \sum_{j=1}^{\frac{x}{\theta}} \frac{|C_j|}{n} \cdot \min\left\{ |C_j|, \sqrt{x/\theta} \right\} 
\geq \gamma^{d_{\text{max}}} \sum_{j:|C_j| \leq \sqrt{x/\theta}} \frac{|C_j|}{n} \cdot \sqrt{x/\theta} \geq \gamma^{d_{\text{max}}} \alpha k \frac{x}{\theta}.
\]

If more than half of rewards in \( S_{\text{OPT}} \) are in CCs such that \( |C_j| \leq \sqrt{x/\theta} \), let \( s \) be the number of such CCs and notice that \( s \leq \frac{x}{\theta} \). We get that:

\[
\text{RA} = \gamma^{d_{\text{max}}} \sum_{j=1}^{\frac{x}{\theta}} \frac{|C_j|}{n} \cdot \min\left\{ |C_j|, \frac{x}{\theta} \right\} 
\geq \gamma^{d_{\text{max}}} \sum_{j:|C_j| \leq \sqrt{x/\theta}} \frac{|C_j|^2}{n} \geq \frac{s}{n} \gamma^{d_{\text{max}}} \left( \frac{1}{s} \sum_{j=1}^{s} |C_j|^2 \right) 
\geq \frac{s}{n} \gamma^{d_{\text{max}}} \left( \frac{1}{s} \sum_{j=1}^{s} |C_j| \right)^2 \geq \gamma^{d_{\text{max}}} \frac{\theta \alpha^2 n}{4x}.
\]
By setting \( \theta = \frac{x}{n^{\frac{2}{3}}} \) we guarantee that the value of RA is at least \( \gamma^{d_{\text{max}}} \alpha^2 n^{1/3}/4 \). Since \( d_{\text{max}} - d_{\text{min}} \leq x \),

\[
\frac{\text{RA}}{\text{OPT}} \geq \frac{\gamma^{d_{\text{max}}} \alpha^2 n^{1/3}/4}{\gamma^{d_{\text{max}}} \alpha n} \geq \frac{\alpha x^2}{4n^{2/3}} = \frac{\alpha}{2n^{2/3}},
\]

where the last inequality follows from the triangle inequality.

**Step 2.** Assume that OPT gets its value from \( n' < n \) rewards that it collects in a segment of length \( x \) (and from all other rewards OPT collects a negligible value). Recall that the NN-RA policy is either NN with probability 0.5 or RA with probability 0.5. By picking the single reward closest to the starting point, NN gets at least \( 1/n' \) of the value of OPT. Otherwise, with probability \( n'/n \), RA starts with one of the \( n' \) rewards picked by OPT and then, by the analysis of step 1, if it sets \( \theta = \frac{x}{(n')^{2/3}} \), RA collects \( \frac{1}{2(n')^{2/3}} \) of the value collected by OPT (we use Step (1) with \( \alpha = 1 \)). It follows that

\[
\frac{\text{NN-RA}}{\text{OPT}} \geq \frac{1}{2} \cdot \frac{1}{n'} + \frac{1}{2} \cdot \frac{n'}{n} \cdot \frac{1}{2(n')^{2/3}} = \frac{1}{2n'} + \frac{(n')^{1/3}}{4n}.
\]

This lower bound is smallest when \( n' \approx n^{3/4} \), in which case NN-RA collects \( \Omega(n^{3/4}) \) of OPT.

**Step 3.** By the same arguments from Step 3 in the analysis of NN-RDFS, it follows that

\[
\frac{\text{NN-RA}}{\text{OPT}} \geq \begin{cases} 
\Omega \left( \frac{n^{-\frac{3}{4}}} {\log(n)} \right), & \text{if } \text{OPT} = \Omega(n) \\
\Omega \left( \frac{n^{-\frac{3}{4}}} {\log(n)} \right), & \text{otherwise.}
\end{cases}
\]

9. **R-NN**

We now analyze the performance guarantees of the R-NN method. The analysis is conducted in two steps. In the first step, we assume that OPT achieved a value of \( \Omega(n') \) by collecting \( n' \) rewards and consider the case that \( n' = \alpha n \). The second step considers the more general case and analyzes the performance of NN-Random for the worst value of \( n' \). We emphasize that unlike the previous two algorithms, we do not assume this time that OPT collects its rewards at a segment of length \( x^{7} \).

**Theorem 5 (R-NN)** For an MDP satisfying Definition 2 with \( n \) rewards:

\[
\frac{\text{R-NN}}{\text{OPT}} \geq \Omega \left( \frac{\log(n)}{n} \right).
\]

**Proof** **Step 1.** Assume OPT collects \( \Omega(n) \) rewards. Define \( x = \log^{\frac{1}{5}}(2) \) and \( \theta = \frac{x}{\sqrt{n}} \) (here we can replace the \( \sqrt{n} \) by any fractional power of \( n \), this will not affect the asymptotics of the result) and denote by \( \{ C_j \} \) the CCs that are obtained by pruning edges longer than \( \theta \). We define a CC to be large if it contains more than \( \log(n) \) rewards. Observe that since there are at most \( \sqrt{n} \) CCs (Lemma 11), at least one large CC exists.

---

7. Therefore, we do not perform a third step like we did in the analysis of the previous methods.
**Lemma 12** Assume that $s_1$ is in a large component $C$. Let $p$ be the path covered by NN starting from $s_1$ until it reaches $s_i$ in a large component. Let $d$ be the length of $p$ and let $r_1$ be the number of rewards collected by NN in $p$ (including the last reward in $p$ which is back in a large component, but not including $s_1$). Note that $r_1 \geq 1$. Then $d \leq (2^{r_1} - 1)\theta$.

**Proof** Let $p_i$ be the prefix of $p$ that ends at the $i^{th}$ reward on $p$ ($i \leq r_1$) and let $d_i$ be the length of $p_i$. Let $\ell_i$ be the distance from the $i^{th}$ reward on $p$ to the $(i+1)^{th}$ reward on $p$. Since when NN is at the $i^{th}$ reward on $p$, the neighbor of $s_1$ in $C$ is at distance at most $d_i + \theta$ from this reward we have that $\ell_i \leq d_i + \theta$. Thus, $d_{i+1} \leq 2d_i + \theta$ (with the initial condition $d_0 = 0$). The solution to this recurrence is $d_i = (2^i - 1)\theta$.

**Lemma 13** For $k < \log(n)$, we have that after $k$ visits of R-NN in large CCs, for any $s$ in a large CC there exists an unvisited reward at distance shorter than $(k+1)\theta$ from $s$.

**Proof** Let $s$ be a reward in a large component $C$. We have collected at most $k$ rewards from $C$. Therefore, there exists a reward $s' \in C$ which we have not collected at distance at most $(k+1)\theta$ from $s$.

Lemma 12 and Lemma 13 imply the following corollary.

**Corollary 14** Assume that $k < \log(n)$, and let $p$ be the path of NN from its $k^{th}$ reward in a large CC to its $(k+1)^{st}$ reward in a large connected component. Let $d$ denote the length of $p$ and $r_k$ be the number of rewards on $p$ (excluding the first and including the last). Then $d \leq (2^{r_k} - 1)(k+1)\theta \leq 2^{r_k+1}k\theta$.

The following lemma concludes the analysis of this step.

**Lemma 15** Let $p$ be the prefix of R-NN of length $x$. Let $k$ be the number of segments on $p$ of R-NN that connect rewards in large CCs and contain internally rewards in small CCs. For $1 \leq i \leq k$, let $r_i$ be the number of rewards R-NN collects in the $i^{th}$ segment. Then $\sum_{i=1}^{k} r_i = \Omega(\log(n))$. (We assume that $p$ splits exactly into $k$ segments, but in fact the last segment may be incomplete, this requires a minor adjustment in the proof.)

**Proof** since $\forall i, r_i \geq 1$, then if $k \geq \log(n)$ the lemma follows. So assume that $s < \log n$. By Corollary 14 we have that

$$x \leq \sum_{i=1}^{k} 2^{r_i+1}i\theta \leq 2^{r_{\text{max}}+2}k^2\theta . \quad (1)$$

where $r_{\text{max}} = \arg\max \{ r_i \}_{i=1}^{k}$. Since $\theta = x/\sqrt{n}$, Equation (1) implies that $\sqrt{n} \leq 2^{r_{\text{max}}+2}k^2$ and since $k \leq \log n$ we get that $\sqrt{n} \leq 2^{r_{\text{max}}+2} \log^2(n)$. Taking logs the lemma follows.

Lemma 15 guarantees that once at $s_1 \in C_j$, R-NN collects $\Omega(\log(n))$ rewards before traversing a distance of $x$. Next, notice that the chance that $s_1$ (as defined in Algorithm 3) belongs to one of the large CCs is $p = \frac{n - \sqrt{n} \log(n)}{n}$, which is larger than $1/2$ for $n \geq 256$. 

18
Finally, similar to NN-RDFS, assume that the value of OPT is greater than a constant fraction of \( n \), i.e., \( \text{OPT} \geq n/2^\alpha \). This means that OPT must have collected the first \( n/2^\alpha + 1 \) rewards after traversing a distance of at most \( \tilde{x} = (\alpha + 1)x \), and denote this fraction of the rewards by \( S_{\text{OPT}} \). Further denote by \( d_{\min}, d_{\max} \) the shortest and longest distances from \( s_0 \) to \( S_{\text{OPT}} \) respectively. By the triangle inequality, \( d_{\max} - d_{\min} \leq \tilde{x} \); therefore, with a constant probability of \( \frac{1}{2^\alpha + 1} \), we get that \( s_1 \in S_{\text{OPT}} \). By taking expectation over the first random pick, it follows that

\[
\frac{\text{R-NN}}{\text{OPT}} \geq \frac{1}{2^\alpha + 1} \cdot \frac{d_{\max} \log(n)}{d_{\min} n} = \frac{\log(n)}{4^\alpha + 1} \geq \Omega \left( \frac{\log(n)}{n} \right).
\]

**Step 2.**

Similar to the analysis of NN-RDFS, we now assume that OPT collects its value from \( n' < n \) rewards that it collects in a segment of length \( x \) (and from all other rewards OPT collects a negligible value). Recall that the R-NN is either NN with probability 0.5 or a random pick with probability 0.5 followed by NN. By picking the single reward closest to the starting point, NN gets at least \( 1/n' \) of the value of OPT. Notice, that we do not need to assume anything about the length of the tour that OPT takes to collect the \( n' \) rewards (since we didn’t use it in Step 1). It follows that:

\[
\frac{\text{R-NN}}{\text{OPT}} \geq \frac{1}{2} \cdot \frac{1}{n'} + \frac{1}{2} \cdot \frac{n'}{n} \cdot \frac{\log(n')}{n'} = \frac{1}{2n'} + \frac{\log(n')}{n}
\]

Thus, in the worst case scenario, \( n' \log(n') \approx n \), which implies that \( n' = \Theta \left( \frac{n}{\log(n)} \right) \). Therefore

\[
\frac{\text{R-NN}}{\text{OPT}} = \Omega \left( \frac{\log(n)}{n} \right).
\]

---

8. To see this, recall that after traversing a distance of \( \tilde{x} \), OPT achieved less than \( n/2^\alpha + 1 \). Since it already traversed \( \tilde{x} \) it can only achieve less than \( n/2^\alpha + 1 \) from the remaining rewards, thus a contraction with the assumption that it achieved more than \( n/2^\alpha \).
10. Exact solutions for the RD-TSP

We now present a variation of the Held-Karp algorithm for the RD-TSP. Note that similar to the TSP, $C(\{S, k\})$ denotes the length of the tour visiting all the cities in $S$, with $k$ being the last one (for TSP this is the length of the shortest tour). However, our formulation required the definition of an additional recursive quantity, $V(\{S, k\})$, that accounts for the value function (the discounted sum of rewards) of the shortest path. Using this notation, we observe that Held-Karp is identical to doing tabular Q-learning on SMDP $M_s$. Since Held-Karp is known to have exponential complexity, it follows that solving $M_s$ using SMDP algorithms is also of exponential complexity.

Algorithm 4 The Held-Karp for the TSP (blue) and RD-TSP (black)

Input: Graph $G$, with $n$ nodes

for $k := 2$ to $n$ do
  $C(\{k\}, k) := d_{1,k}$
  $C(\{k\}, k) := d_{1,k}$
  $V(\{k\}, k) := \gamma^{d_{1,k}}$
end for

for $s := 2$ to $n - 1$ do
  for all $S \subseteq \{2, ..., n\}, |S| = s$ do
    for all $k \in S$ do
      $C(S, k) = \min_{m \neq k, m \in S}[C(S \setminus \{k\}, m) + d_{m,k}]$
      $Q(S, k, a) = [V(S \setminus \{k\}, a) + \gamma^{C(S \setminus \{k\}, a)} \cdot \gamma^{d_{a,k}}]$
      $a^* = \arg\max_{a \neq k, a \in S} Q(S, k, a)$
      $C(S, k) = C(S \setminus \{k\}, a^*) + d_{a^*,k}$
      $V(S, k) = Q(S, k, a^*)$
    end for
  end for
end for

$opt := \min_{k \neq 1}[C(\{2, ..., n\}, k) + d_{k,1}]$

$opt := \max_{k \neq 1}[V(\{2, ..., n\}, k) + \gamma^{C(\{2, ..., n\}, k)} + d_{k,1}]$

return $(opt)$

10.1 Exact solutions for simple geometries

We now provide exact, polynomial time solutions based on dynamic programming for simple geometries, like a line and a star. We note that such solutions (exact and polynomial) cannot be derived for general geometries.

10.1.1 Dynamic programming on a line (1D)

Given an RD-TSP instance, such that all the rewards are located on a single line (denoted by the integers $1, \ldots, n$ from left to right), it is easy to see that an optimal policy collects, at any time, either the nearest reward to the right or the left of its current location. Thus, at any time the set of rewards that it has already collected lie in a continuous interval between the first uncollected reward to the left of the origin, denoted $\ell$, and the first uncollected reward to the right of the origin denoted $r$. The action to take next is either to collect $\ell$ or to collect $r$. 
It follows that the state of the optimal agent is uniquely defined by a triple \((\ell, r, c)\), where \(c\) is the current location of the agent. Observe that \(c \in \{\ell + 1, r - 1\}\) and therefore there are \(O(n^2)\) possible states in which the optimal policy could be.

Since we were able to classify a state space of polynomial size which contains all states of the optimal policy, then we can describe a dynamic programming scheme (Algorithm 5) that finds the optimal policy. The algorithm computes a table \(V\), where \(V(\ell, r, \to)\) is the maximum value we can get by collecting all rewards \(1, \ldots, \ell\) and \(r, \ldots, n\) starting from \(r - 1\), and \(V(\ell, r, \leftarrow)\) is defined analogously starting from \(\ell + 1\). The algorithm first initializes the entries of \(V\) where either \(\ell = 0\) or \(r = n + 1\). These entries correspond to the cases where all the rewards to the left (right) of the agent have been collected. (In these cases the agent continues to collect all remaining rewards one by one in their order.) It then iterates over \(t\), a counter over the number of rewards that are left to collect. For each value of \(t\), we define \(S\) as all the combinations of partitioning these \(t\) rewards to the right and the left of the agent. We fill \(V\) by increasing value of \(t\). To fill an entry \(V(\ell, r, \leftarrow)\) such that \(\ell + (n + 1 - r) = t\) we take the largest among 1) the value to collect \(\ell\) and then the rewards \(1, \ldots, \ell - 1\) and \(r, \ldots, n\), appropriately discounted and 2) the value to collect \(r\) and then \(1, \ldots, \ell\) and \(r + 1, \ldots, n\). We fill \(V(\ell, r, \to)\) analogously.

The optimal value for starting position \(j\) is \(1 + V(j - 1, j + 1, \to)\). Note that the Algorithm computes the value function; to get the policy, one has merely to track the argmax at each maximization step.

### 10.1.2 Dynamic Programming on a \(d\)-Star

We consider an RD-TSP instance, such that all the rewards are located on a \(d\)-star, i.e., all the rewards are connected to a central connection point via one of \(d\) lines and there are \(n_i\) rewards along the \(i\)th line. We denote the rewards on the \(i\)th line by \(m_i^j \in \{1, \ldots, n_i\}\), ordered from the origin to the end of the line, and focus on the case where the agent starts at the origin.\(^9\) It is easy to see that an optimal policy collects, at any time, the uncollected reward that is nearest to the origin along one of the \(d\) lines. Thus, at any time the set of rewards that has already collected lie in \(d\) continuous intervals between the origin and the first uncollected reward along each line, denoted by \(\ell = \{\ell_i\}_{i=1}^d\). The action to take next is to collect one of these nearest uncollected rewards. It follows that the state of the optimal agent is uniquely defined by a tuple \((\ell, c)\), where \(c\) is the current location of the agent. Observe that \(c \in \{m_i^{\ell_i-1}\}_{i=1}^d\) and therefore there are \(O(dn^d)\) possible states in which the optimal policy could be.

Since we were able to classify a state space of polynomial size which contains all states of the optimal policy then we can describe a dynamic programming scheme (Algorithm 7) that finds the optimal policy. The algorithm computes a table \(V\), where \(V(\ell, c)\) is the maximum value we can get by collecting all rewards \(\{m_{\ell_i}^1, \ldots, m_{\ell_i}^{d}\}_{i=1}^d\) starting from \(c\). The algorithm first initializes the entries of \(V\) where all \(\ell_i = n_i + 1\) except for exactly one entry. These entries correspond to the cases where all the rewards have been collected, except in one line segment (in these cases the agent continues to collect all remaining rewards one by one in their order.) It then iterates over \(t\), a counter over the number of rewards that are left to collect. For each value of \(t\), we define \(S\) as all the combinations of partitioning these \(t\) rewards among \(d\) lines. We fill \(V\) by increasing value of \(t\). To fill an entry \(V(\ell, c)\) such that \(\sum l_i = n - t\) we take the largest among the values for collecting \(\ell_i\) and then the rewards \(m_1^{\ell_1}, \ldots, m_{\ell_{i+1}}^{i}, \ldots, m_n^{\ell_{i}}\) appropriately discounted.

\(^9\) The more general case is solved by applying Algorithm 5 until the origin is reached followed by Algorithm 7
Algorithm 6 Optimal solution for the RD-TSP on a line. The rewards are denoted by 1,\ldots,n from left to right. We denote by \(d_{i,j}\) the distance between reward \(i\) and reward \(j\). We denote by \(V(\ell, r, \rightarrow)\) the maximum value we can get by collecting all rewards 1,\ldots,\(\ell\) and \(r,\ldots,n\) starting from reward \(r - 1\). Similarly, we denote by \(V(\ell, r, \leftarrow)\) maximum value we can get by collecting all rewards 1,\ldots,\(\ell\) and \(r,\ldots,n\) starting from \(\ell + 1\). If the leftmost (rightmost) reward was collected we define \(\ell = 0\) (\(r = n + 1\)).

**Input:** Graph \(G\), with \(n\) nodes

Init \(V(\cdot, \cdot, \cdot) = 0\)

for \(t = 1, n\) do

\[
V(\ell = t, n + 1, \rightarrow) := \gamma^{d_{t,n}} \cdot \left(1 + \sum_{j=t}^{n-1} \gamma \sum_{i=0}^{j} d_{i,i+1}\right)
\]

\[
V(0, r = t, \leftarrow) := \gamma^{d_{1,t}} \cdot \left(1 + \sum_{j=1}^{n-1} \gamma \sum_{i=0}^{j} d_{i,i+1}\right)
\]

end for

for \(t = 2, \ldots n - 1\) do

\(S = \{(i, n + 1 - j)|i + j = t\}\)

for \((\ell, r) \in S\) do

if \(V(\ell, r, \rightarrow) = 0\) then

\[
V(\ell, r, \leftarrow) = \max \left\{\gamma^{d_{\ell,\ell+1}} [1 + V(\ell - 1, r, \leftarrow)] , \gamma^{d_{\ell+1,r}} [1 + V(\ell, r + 1, \rightarrow)]\right\}
\]

end if

if \(V(\ell, r, \leftarrow) = 0\) then

\[
V(\ell, r, \rightarrow) = \max \left\{\gamma^{d_{\ell,r-1}} [1 + V(\ell - 1, r, \leftarrow)] , \gamma^{d_{r-1,r}} [1 + V(\ell, r + 1, \rightarrow)]\right\}
\]

end if

end for

end for

Note that the Algorithm computes the value function; to get the policy, one has merely to track the argmax at each maximization step.
Algorithm 8 Optimal solution for the RD-TSP on a d-star. We denote by $n_i$ the amount of rewards there is to collect on the $i$th line, and denote by $m_j^t \in \{1,..,n_i\}$ the rewards along this line, from the center of the star to the end of that line. We denote by $d_{m_j^t,m_j^k}$ the distance between reward $i$ on line $t$ and reward $j$ on line $k$. The first uncollected reward along each line is denoted by $\ell_i$, and the maximum value we can get by collecting all the remaining rewards $m_1^{\ell_i},...,m_{n_i}^{\ell_i},...,m_d^{\ell_i},...,m_d^{n_i}$ starting from reward $c$ is defined by $V(\bar{\ell} = \{\ell_i\}_{i=1}^d, c)$. If all the rewards were collected on line $i$ we define $\ell_i = n_i + 1$.

Input: Graph $G$, with $n$ rewards.
Init $V(\cdot,\cdot) = 0$
for $i \in \{1,..,d\}$ do
  for $\ell_i \in \{1,..,n_i\}$ do
    for $c \in \{m_1^{\ell_i},..,m_{\ell_i-1}^{\ell_i},m_{\ell_i}^{\ell_i}\}$ do
      $V(n_1+1,..,\ell_i, n_d+1, c) = \gamma^{d_{c,m_{\ell_i}}} \cdot \left(1 + \sum_{j=\ell_i}^{n_i} \gamma \sum_{k=0}^{d_{k,k+1}} \right)$
    end for
  end for
for $t = 2,..n-1$ do
  $S = \{\bar{\ell}|\ell_i \in \{1,..,n_i\}\sum \ell_i = n-t\}$
  for $\bar{\ell} \in S, c \in \{m_1^{\ell_i-1}\}_{i=1}^d$ do
    if $V(\bar{\ell},c) = 0$ then
      $A = \{m_j^i | j = \ell_i, j \leq n_i\}$
      for $a \in A$ do
        $V(\bar{\ell},c) = \max \left\{ V(\bar{\ell},c), \gamma^{d_{c,a}} \left[1 + V(\bar{\ell}+e_a,a)\right] \right\}$
      end for
    end if
  end for
end for
11. Deterministic Learning Simulations

In this section, we provide additional learning simulations for a deterministic MDP. The setting and the conclusions follow the text described in the main paper.

Figure 1: **Top, Left:** Grid-world environment, where an agent (yellow) collects rewards (teal) in a maze (walls in red). **Top, Right:** Evaluation of options success % (red) and option time gap (blue) during learning. **Bottom:** The performance of local policies during learning.

12. Planning Simulations

In this section, we evaluate and compare the performance of deterministic and stochastic local policies by measuring the (cumulative discounted) reward achieved by each algorithm on different MDPs as a function of $n$, the number of the rewards, with $n \in \{100, 200, 400, 600, 800, 1000\}$. For each MDP, the algorithm is provided with the set of optimal options $\{o_i\}_{i=1}^n$ and their corresponding value functions. We are interested to test our algorithms in the regime where OPT can collect almost all of the rewards within a constant discount (i.e., $OPT \approx \alpha n$), but, there also exist bad tours that achieve a constant value (i.e., taking the most distance reward in each step); thus, we set $\gamma = 1 - \frac{1}{n}$. 


We always place the initial state $s_0$ at the origin, i.e., $s_0 = (0, 0)$. We define $x = \log_2(2)$, and $\ell = 0.01x$ denotes a short distance.

Next, we describe five MDP types (Figure 3, ordered from left to right) that we considered for evaluation. For each of these MDP types, we generate $N_{MDP} = 10$ different MDPs, and report the average reward achieved by each algorithm (Figure 2, Top), and in the worst-case (the minimal among the $N_{MDP}$ scores) (Figure 2, Bottom). As some of our algorithms are stochastic, we report average results, i.e., for each MDP we run each algorithm $N_{alg} = 100$ times and report the average score.

Figure 3 visualizes these MDPs, for $n = 800$ rewards, where each reward is displayed on a 2D grid using gray dots. For each MDP type, we present a single MDP sampled from the appropriate distribution. For the stochastic algorithms, we present the best (Figure 4) and the worst tours (Figure 3), among 20 different runs (for NN we display the same tour since it is deterministic). Finally, for better interpretability, we only display the first $n/k$ rewards of each tour, in which the policy collects most of its value, with $k = 8$ ($n/k = 100$) unless mentioned otherwise.

**Figure 2**: Evaluation of deterministic and stochastic local policies over different MDPs. The cumulative discounted reward of each policy is reported for the average and worst case scenarios.

(1) **Random Cities.** For a vanilla TSP with $n$ rewards (nodes) randomly distributed on a 2D plane, it is known that the NN algorithm yields a tour which is 25% longer than optimal on average (Johnson and McGeoch, 1997). We used a similar input to compare our algorithms, specifically, we generated an MDP with $n$ rewards $r_i \sim (U(0, x), U(0, x))$, where $U$ is the uniform distribution.

Figure 2 (left), presents the results for such MDP, we can see that the NN algorithm performs the best both on the average and in the worst case. This observation suggests that when the rewards are distributed at random, selecting the nearest reward is a reasonable thing to do. In addition, we can
see that NN-RDFS performs the best among the stochastic policies (as predicted by our theoretical results). On the other hand, the RA policy performs the worst among stochastic policies. This happens because sorting the rewards by their distances from \( s_1 \), introduces an undesired “zig-zag” behavior while collecting rewards at equal distance from \( s_1 \) (Figure 3).

(2) Line. This MDP demonstrates a scenario where greedy algorithms like NN and R-NN are likely to fail. The rewards are located in three different groups; each contains \( n/3 \) of the rewards. In group 1, the rewards are located in a cluster left to the origin \( r_i \sim (U[-\theta/3 - \ell, -\theta/3 + \ell], N(0, \ell)) \), while in group 2 they are located in a cluster right to the origin but a bit closer than group 1 \( r_i \sim (U[\theta/3 - 3\ell, \theta/3 - 2\ell], N(0, \ell)) \). Group 3 is also located to the right, but the rewards are placed in increasing distances, such that the \( i \)-th reward is located at \((\theta/3)2^i\).

For visualization purposes, we added a small variance in the locations of the rewards at groups 1 and 2 and rescaled the axes. The two vertical lines of rewards represent these two groups, while we cropped the graph such that only the first few rewards in group 3 are observed. Finally, we chose \( k = 2 \), such the first half of the tour is displayed, and we can see the first two groups visited in each tour.

Inspecting the results, we can see that NN and R-NN indeed perform the worst. To understand this, consider the tour that each algorithm takes. NN goes to group 2, then 3 then 1 (and loses a lot from going to 3). The stochastic tours depend on the choice of \( s_1 \). If it belongs to group 1, they collect group1 then 2 then 3, from left to right, and perform relatively the same. If it belongs to group 3, they will first collect the rewards to the left of \( s_1 \) in ascending order and then come back to collect the remaining rewards to the right, performing relatively the same. However, if \( s_1 \) is in group 2, then NN-RDFS, NN-RA will visit group 1 before going to 3, while R-NN is tempted to go to group 3 before going to 1 (and loses a lot from doing it).

(3) Random Clusters. This MDP demonstrates the advantage of stochastic policies. We first randomly place \( k = 10 \) cluster centers \( c_j^i \), \( j = 1, \ldots, k \) on a circle of radius \( x \). Then to draw a reward \( r_i \) we first draw a cluster center \( c_j^i \) uniformly and then draw \( r_i \) such that \( r_i \sim (U[c^i_j - 10\ell, c^i_j + 10\ell], U[c^j_y - 10\ell, c^j_y + 10\ell]) \).

This scenario is motivated by maze navigation problems, where collectible rewards are located at rooms (clusters) while in between rooms there are fewer rewards to collect (similar to Figure 4). Inspecting the results, we can see that NN-RDFS and R-NN perform the best, in particular in the worst case scenario. The reason for this is that NN picks the nearest reward, and most of its value comes from rewards collected at this cluster. On the other hand, the stochastic algorithms visit larger clusters first with higher probability and achieve higher value by doing so.

(4) Circle. In this MDP, there are \( \sqrt{n} \) circles, all centered at the origin, and the radii of the \( i \)th circle is \( \rho_i = \frac{x}{\sqrt{n}} \cdot (1 + \frac{1}{\sqrt{n}})^i \). On each circle we place \( \sqrt{n} \) rewards place at equal distances. This implies that the distance between adjacent rewards on the same circle is longer than the distance between adjacent rewards on two consecutive circles.

Examining the tours, we can see that indeed NN and R-NN are taking tours that lead them to the outer circles. On the other hand, RDFS and RA are staying closer to the origin. Such local behavior is beneficial for RDFS, which achieves the best performance in this scenario. However, while RA performs well in the best case, its performance is much worse than the other algorithms in the worst case. Hence, its average performance is the worst in this scenario.
Figure 3: Visualization of the worst tours, taken by deterministic and stochastic local policies for $n = 800$ rewards.
(5) Rural vs. Urban. Here, the rewards are sampled from a mixture of two normal distributions. Half of the rewards are located in a “city”, i.e., their position is a Gaussian random variable with a small standard deviation s.t. \( r_i \sim (N(x, \ell), N(0, \ell)) \); the other half is located in a “village”, i.e., their position is a Gaussian random variable with a larger standard deviation s.t. \( r_i \sim (N(-x, 10x), N(0, 10x)) \). To improve the visualization here, we chose \( k = 2 \), such the first half of the tour is displayed. Since half of the rewards belong to the city, choosing \( k = 2 \) ensures that any tour that is reaching the city only the first segment of the tour (until the tour reaches the city) will be displayed.

In this MDP, we can see that in the worst case scenario, the stochastic policies perform much better than NN. This happens because NN is mistakenly choosing rewards that take it to remote places in the rural area, while the stochastic algorithms remain near the city with high probability and collect its rewards.

13. Related work

Pre-defined rules for option selection are used in several studies. Karlsson (1997) suggested a policy that chooses greedily with respect to the sum of the local Q-values \( a^* = \arg\max_a \sum_i Q_i(s, a) \). Humphrys (1996) suggested to choose the option with the highest local Q-value \( a^* = \arg\max_{a,i} Q_i(s, a) \) (NN). Such greedy combination of local policies that were optimized separately may not necessarily perform well. Barreto et al. (2017) considered a transfer framework similar to ours, but did not focus
on collectible reward decomposition (Definition 2). Instead, they proposed a framework where the rewards are linear in some reward features \( R_i(s, a) = w_i^T \phi(s, a) \). Similar to (Humphrys, 1996), they suggested using NN as the pre-defined rule for option selection (but referred to it as GPI). In addition, the authors provided performance guarantees for using GPI in the form of additive (based on regret) error bounds but did not provide impossibility results. In contrast, we prove multiplicative performance guarantees for NN, as well as for three stochastic policies. We also proved, for the first time, impossibility results for such local option selection methods.

A different approach to tackle these challenges is **Multi-task learning**, in which we optimize the options in parallel with the policy over options (Russell and Zimdars, 2003; Sprague and Ballard, 2003; van Seijen et al., 2017). One method that achieves that goal is the local SARSA algorithm (Russell and Zimdars, 2003; Sprague and Ballard, 2003). Similar to (Karlsson, 1997), a Q function is learned locally for each option (concerning a local reward). However, here the local Q functions are learnt on-policy (using SARSA) with respect to the policy over options \( \pi(s) = \arg\max_a \sum_i Q_i(s, a) \), instead of being learned off-policy with Q learning. Russell and Zimdars (2003) showed that if the policy over options is being updated in parallel with the local SARSA updates, then the local SARSA algorithm is promised to converge to the optimal value function.

---

10. Notice that Definition 2 is a special case of the framework considered by Barreto et al. (2017).