Efficient Value Propagation with the Compositional Optimality Equation

Piotr Piękos[1,2], **Aditya Ramesh**[2], **Francesco Faccio**[1,2], **Jürgen Schmidhuber**[1,2]
[1]AI Initiative, KAUST, Thuwal, Saudi Arabia
[2]The Swiss AI Lab, IDSIA, USI & SUPSI, Lugano, Switzerland
{piotr.piekos, juergen.schmidhuber}@kaust.edu.sa
{aditya.ramesh, francesco.faccio}@idsia.ch

Abstract

Goal-Conditioned Reinforcement Learning (GCRL) is about learning to reach predefined goal states. GCRL in the real world is crucial for adaptive robotics. Existing GCRL methods, however, suffer from low sample efficiency and high cost of collecting real-world data. Here we introduce the Compositional Optimality Equation (COE) for a widely used class of deterministic environments in which the reward is obtained only upon reaching a goal state. COE represents a novel alternative to the standard Bellman Optimality Equation, leading to more sample-efficient update rules. The Bellman update combines the immediate reward and the bootstrapped estimate of the best next state. Our COE-based update rule, however, combines the best composition of two bootstrapped estimates reflecting an arbitrary intermediate subgoal state. In tabular settings, the new update rule guarantees convergence to the optimal value function exponentially faster than the Bellman update! COE can also be used to derive compositional variants of conventional (deep) RL. In particular, our COE-based version of DDPG is more sample-efficient than DDPG in a continuous grid world.

1 Introduction

Reinforcement Learning (RL) [43] in the real world (for robotics, etc.) remains difficult [40]. A major obstacle is the low sample efficiency of most RL algorithms, compounded by the high cost of data acquisition. Here, we focus on goal-conditional RL (GCRL), where a reward is received upon reaching a specified goal given in advance [15, 32, 33].

When goals are distant and widely distributed across the space [12, 14, 27], credit for reaching them should be rapidly assigned across long-time lags to relevant, previously executed actions. To achieve this, multi-step methods estimate the value after unrolling the trajectory for a chosen number of steps. [30, 38] These methods, however, have been show to suffer from high variance [5].

Multi-step methods construct value targets from rewards observed during rollout and the estimated value of the state achieved at the end of the rollout. This can be seen as a combination of Monte-Carlo estimate achieved through the rollout with bootstrapping in estimating the value of the last achieved state. Here, we propose an alternative approach that uses two bootstrapped estimated values–from the current state to a subgoal, and from the subgoal to the goal–to design updates.

Specifically, we derive a method tailored to GCRL that improves sample efficiency and enables **exponentially** faster convergence than the traditional Bellman update for a class of deterministic tabular environments. It injects hierarchical inductive bias into the value function to utilize long-distance information in the update. We extend this result to model-free algorithms with function approximators and empirically demonstrate its convergence in a continuous gridworld environment.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2 Method

Smart decomposition of large problems into smaller subproblems is the foundation of many efficient algorithms. Here we investigate the classic Bellman Optimality Equation and propose a more efficient, adaptive decomposition method designed specifically for a large class of GCRL problems. The augmented state space $\mathcal{S} = \mathcal{S}_S \times \mathcal{S}_G$ is defined as the cartesian product of the original state space $\mathcal{S}_S$ of the environment and the goal space $\mathcal{S}_G$. Here, we assume that the original state space is the same as the goal space: $\mathcal{S}_S = \mathcal{S}_G$.

An agent interacts with a Markov Decision Process (MDP) by taking actions $A_t \in \mathcal{A}$ based on the current state $S_t \in \mathcal{S}_S$ and goal $G \in \mathcal{S}_G$. Concretely, actions are sampled from a policy that maps states and goals to a distribution over actions $\pi : \mathcal{S}_S \times \mathcal{S}_G \to \Delta(\mathcal{A})$. In response to the agent's action, the MDP provides a new state S_{t+1}, sampled from the distribution over states output by a transition function $\mathcal{T} : \mathcal{S}_S \times \mathcal{A} \to \Delta(\mathcal{S}_S)$. The MDP also provides rewards R_{t+1} based on the state and goal. We assume that the agent receives a reward from s to g equal to $r(s, g) = \gamma \mathbf{1}_{\{s=g\}}$, where $\gamma \in (0, 1]$ is a discount factor. The optimal value function from state to goal $V^*(s, g) = \max_\pi \mathbf{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} | S_t = s, G = g]$ is the highest expected cumulative discounted sum of rewards that the agent can obtain from state s. Analogously, we consider action-value functions and optimal action-value functions, $Q^*(s, a, g) = \max_\pi \mathbf{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} | S_t = s, A_t = a, G = g]$. Under these assumptions, we introduce our **Compositional Optimality Equation (COE):**

$$V^*(s, g) = \max_{subg \in \mathcal{S}_S} V^*(s, subg) V^*(subg, g).$$

The equation composes two value functions using the information from a subgoal state. This equation is formally motivated(App. B.1) and is satisfied by the optimal policy(App. B.1). From the COE, we derive popular COE-based versions of RL algorithms such as COE Value Iteration (COE-VI) (App. 2.1) or Deep RL algorithms COE-DDPG and COE-DQN (App. D). Moreover, we prove that COE-VI **converges in a logarithmic number of steps in terms of distance to the goal** (App. B.1.1).

2.1 COE-Value Iteration

The new optimality equation enables the definition of a new Compositional update rule for value iteration. Here we consider a deterministic transition function $\mathcal{T} : \mathcal{S}_S \times \mathcal{A} \to \mathcal{S}_S$.

Algorithm 1 Generalized Compositional Value Iteration

1: Initialize $V = 0$
2: **for** $s \in S, a \in \mathcal{A}$ **do**
3: $\quad V(s, \mathcal{T}(s, a)) = \gamma$
4: $\quad V(s, s) = 1$
5: **end for**
6: **repeat**
7: $\quad$ **for** $s \in S, g \in S$ **do**
8: $\quad\quad subg \leftarrow$ ProduceSubgoal
9: $\quad\quad V_2 \leftarrow V(s, subg) V(subg, g)$
10: $\quad$ **end for**
11: $\quad V \leftarrow V_2$
12: **until** convergence

Value iteration as a special case of Generalized Compositional Value Iteration If ProduceSubgoal yields $subg \leftarrow \arg\max_{a \in \mathcal{A}} V(s, \mathcal{T}(s, a)) V(\mathcal{T}(s, a), g)$ then it is equivalent to the standard Bellman-based Value Iteration.

COE-Value iteration COE-Value Iteration (COE-VI) is achieved when for ProduceSubgoal we select $subg \leftarrow \arg\max_{subg \in S} V(s, subg) V(subg, g)$. Therefore, for COE-Value Iteration, lines 6 and 7 can be rewritten shortly as $V_2 \leftarrow \max_{subg \in S} V(s, subg) V(subg, g)$.

For COE-VI, it is necessary that the value estimates are initialized with values smaller than the optimal values because $V(s, g)$ can not decrease after an iteration of COE-VI. As a consequence, apart from the values that we a priori know, we initialize the value function estimate with zeros.

3 Generalization to Deep RL

Analogously to how popular Deep RL algorithms are derived from the Bellman Optimality Equation, it is possible to derive COE-based equivalents. In this section, we describe the process of obtaining deep RL algorithms from our tabular COE Value Iteration algorithm (refer to algorithm 1). We provide examples with pseudocode for COE-DDPG and COE-DQN, detailed further in appendix D.

3.1 Action-value function

Transitioning from a planning approach to a model-free learning algorithm requires adapting COE into a state-action value function update capable of directly utilizing experience samples.

The definition of optimal trajectories can be subtle. In the expression $Q(s, a, g)$, committing to the action a might lead to inconsistencies if the discovered subgoal that maximizes $V(s, subg)V(subg, g)$ does not entail selecting action a. Therefore, it is necessary to use optimal trajectories from state s to goal g that select a as the initial action. In deterministic model-based setups, this expression can be simplified with $V(s', subg)$, where $s' = \mathcal{T}(s, a)$. Nonetheless, to ensure our update remains model-free, we condition our subgoal generator function, ProduceSubgoal, on action a as well. The new declaration becomes ProduceSubgoal: $S \times \mathcal{A} \times S \to S$:

$$Q(s, a, g) \leftarrow Q(s, a, subg) \max_{a_2} Q(subg, a_2, g) \text{ for } subg \text{ in optimal trajectories from } s' \text{ to } g$$

3.2 Subgoal Generator

The main difference between Bellman-based algorithms and COE-based algorithms is their bootstrapped estimator. Traditional algorithms following the Bellman style decompose rewards into immediate next steps combined with an estimate of the remaining rewards. Contrarily, COE-based algorithms utilize two bootstrapped estimates of subproblem values, as described above.

A critical aspect of the algorithm is the decomposition of the problem. Specifically, selecting an appropriate subgoal. In gridworld scenarios, this selection is guided by the maximization of the product of value estimates. The state that maximizes this product is chosen as the subgoal, and its estimate is subsequently used for the update. However, in continuous state environments, attempting direct maximization is infeasible due to the extensive nature of the space, which could lead to an extreme overestimation bias, as mentioned by Van Hasselt et al. [45]. To tackle this, we introduce a separate neural network aimed at predicting the state where the product of value functions is maximized.

We call this network the Generator, denoted as $G : \mathcal{S}_\mathcal{S} \times \mathcal{A} \times \mathcal{S}_\mathcal{G} \to \mathcal{S}_\mathcal{S}$, as it generates a subgoal state based on the current state, action and goal. The Generator is parametrized by θ_g. It is trained by maximizing the state-action version of the product of values defined in COE 2. Therefore, at each step, the generator is updated via gradient ascent according to the following gradient expression:

$$\nabla_{\theta_g} \frac{1}{|B|} \sum_{(s,a,r,s',g,d) \in B} (1 - d)Q(s', a, subg)Q(subg, \pi(subg, g), g) \text{ where } subg = G(s, a, g),$$

where d is the flag signalizing whether s' is terminal.

3.3 Grounding update

In the Algorithm 1, the initialization with $V(s, \mathcal{T}(s, a)) = \gamma$ is crucial for the algorithm to work. It actually gives the algorithm information about the environment dynamics as well as works as a basis for the induction from which all combined values between pairs of states and goals are derived in the proof B.1.1.

To mimic this basis for composition in deep RL, we create the so-called "Grounding Update". We train the value function to equal γ for the transitions from the experience replay buffer with the next state as a goal. Therefore, at each step, apart from the standard COE update, we also perform the

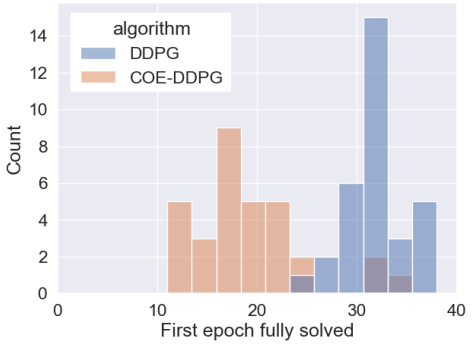

Figure 1: Histogram of first epochs that solved continuous gridworld. COE-DDPG converges significantly faster than DDPG.

Dist. to the goal	#steps to Opt. Value		#steps to Opt. Policy	
	VI	COE-VI	VI	COE-VI
Empirical results				
2	2	1	2	1
10	10	4	6.33	4
25	25	5	19.8	5
50	50	**6**	38.14	**6**
Theoretical guarantees				
100	100	7	100	7
1000	1000	10	1000	10
10000	10000	**14**	10000	**14**

Table 1: Comparison of COE-based Value Iteration (COE-VI) to Value Iteration (VI) in terms of learning speed on the gridworld environment.

grounding update by taking a step in the direction of descent with the following gradient:

$$\nabla_{\theta_q} \frac{1}{|B|} \sum_{(s,a,r,s',g,d)\in B} (Q(s,a,s') - \gamma),$$

where θ_q is the parametrization of Q.

Note, that for the grounding update, we don't use the goal from the buffer, but we treat the next state s' as a goal, so we push all direct transitions to have a value γ in the value function.

4 Results

In this section, we describe the experimental setup used to demonstrate the empirical results of our algorithm, together with the implementation details. We compare the sample efficiency of *COE Value iteration* with *Bellman Value Iteration*, and *COE-DDPG* with DDPG. We evaluate them on gridworld environments - discrete gridworld for value iteration and continuous gridworld for DDPG. Our results in both settings show that our COE-based algorithms are superior to their Bellman counterparts in terms of the number of updates required to solve the environment.

We evaluate COE-VI on 2-dimensional gridworld environments, each composed of 50×50 tiles. For each board, we choose with probability 0.3 whether each tile should be a wall or a floor. Next, we randomly sample the start state and the goal state from all possible floors and check whether the goal is reachable from the start state. The results in Figure 1 are consistent with the theoretical derivations.

To understand COE's generalization to Deep RL, we also evaluate COE-DDPG in the continuous grid world environment and compare it to standard Bellman-based DDPG. We evaluate how quickly both algorithms achieve an entire epoch where all goals are reached within a specific time frame. We repeat this 32 times to obtain a measure of sample efficiency. The histogram of the first epochs solved in each run is shown in Figure 1. COE-DDPG, on average, requires 40.6% fewer epochs than DDPG. Details of both experiments and an extended discussion of the results can be found in the (App. C).

5 Summary

Our novel Compositional Optimality Equation (COE) allows for Reinforcement Learning based on value decomposition, yielding algorithms significantly more sample-efficient than the traditional Bellman update.

Our experiments confirm COE's significant speed-ups. The remaining limitations include: (1) we assume identical state and goal spaces; (2) our theoretical derivation is restricted to deterministic MDPs; (3) there are no results for partially observable environments; (4) the direct operation on states becomes infeasible in higher dimensions. Extending COE's applicability to real-world environments is a promising direction for future research.

References

[1] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup. A survey of exploration methods in reinforcement learning. *arXiv preprint arXiv:2109.00157*, 2021.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[3] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[5] C. Cortes, Y. Mansour, and M. Mohri. Learning bounds for importance weighting. *Advances in neural information processing systems*, 23, 2010.

[6] P. Dayan and G. Hinton. Feudal reinforcement learning. In D. S. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (NIPS) 5*, pages 271–278. Morgan Kaufmann, 1993.

[7] R. Dearden, N. Friedman, and S. Russell. Bayesian q-learning. *Aaai/iaai*, 1998:761–768, 1998.

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

[9] F. Faccio, A. Ramesh, V. Herrmann, J. Harb, and J. Schmidhuber. General policy evaluation and improvement by learning to identify few but crucial states. *arXiv preprint arXiv:2207.01566*, 2022.

[10] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[11] W. Guo, X. Wu, U. Khan, and X. Xing. Edge: Explaining deep reinforcement learning policies. *Advances in Neural Information Processing Systems*, 34:12222–12236, 2021.

[12] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[13] A. Harutyunyan, M. G. Bellemare, T. Stepleton, and R. Munos. Q () with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

[14] C. Hoang, S. Sohn, J. Choi, W. Carvalho, and H. Lee. Successor feature landmarks for long-horizon goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 34:26963–26975, 2021.

[15] L. P. Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pages 1094–8. Citeseer, 1993.

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

[17] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[19] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

[20] H. Liu, M. Zhuge, B. Li, Y. Wang, F. Faccio, B. Ghanem, and J. Schmidhuber. Learning to identify critical states for reinforcement learning from videos. *arXiv preprint arXiv:2308.07795*, 2023.

[21] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

[22] V. Mai, K. Mani, and L. Paull. Sample efficient deep reinforcement learning via uncertainty estimation. *arXiv preprint arXiv:2201.01666*, 2022.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[24] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare. Safe and efficient off-policy reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[25] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[26] X. Pan, Y. You, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[27] S. Park, D. Ghosh, B. Eysenbach, and S. Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *arXiv preprint arXiv:2307.11949*, 2023.

[28] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10, 1997.

[29] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[30] D. Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[31] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

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

[33] J. Schmidhuber. Reinforcement learning in markovian and non-markovian environments. *Advances in neural information processing systems*, 3, 1990.

[34] J. Schmidhuber. *Towards compositional learning with dynamic neural networks*. Inst. für Informatik, 1990.

[35] J. Schmidhuber. Adaptive history compression for learning to divide and conquer. In *Proc. International Joint Conference on Neural Networks*, pages 1130–1135, 1991.

[36] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227, 1991.

[37] J. Schmidhuber and R. Wahnsiedler. Planning simple trajectories using neural subgoal generators. *From Animals to Animats*, 2:196, 1993.

[38] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[39] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[40] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, et al. The limits and potentials of deep learning for robotics. *The International journal of robotics research*, 37(4-5):405–420, 2018.

[41] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3: 9–44, 1988.

[42] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

[43] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[44] S. B. Thrun. Efficient exploration in reinforcement learning. 1992.

[45] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[46] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[47] Y. Wang, H. Liu, M. Strupl, F. Faccio, Q. Wu, X. Tan, and J. Schmidhuber. Highway reinforcement learning. 2022.

[48] M. Wiering and J. Schmidhuber. Hq-learning. *Adaptive behavior*, 6(2):219–246, 1997.

[49] M. Zawalski, M. Tyrolski, K. Czechowski, D. Stachura, P. Piękos, T. Odrzygóźdź, Y. Wu, Ł. Kuciński, and P. Miłoś. Fast and precise: Adjusting planning horizon with adaptive subgoal search. *arXiv preprint arXiv:2206.00702*, 2022.

A Related Work

Deep Reinforcement Learning (RL)[43] has emerged as a powerful paradigm for training intelligent agents across a wide variety of applications, ranging from gaming [23, 39, 46] to natural language processing[4, 25] and robotics [17, 29, 31]. However, training Reinforcement learning agents continues to be a challenging endeavor due to its substantial energy consumption and intensive computational requirements. These challenges amplify when RL is applied in real-life scenarios, where each sample is considerably more costly compared to simulated samples [21, 26, 40].

To alleviate the training costs for real-life robots and RL agents, a series of research papers have proposed strategies to improve sample efficiency. Dearden et al. [7], Mai et al. [22] leverage uncertainty to better utilize information from the value functions, while Arjona-Medina et al. [3], Faccio et al. [9], Guo et al. [11], Liu et al. [20] tackle the credit assignment problem by explicitly identifying crucial states. Wang et al. [47] propose an alternative to the Bellman Equation, aiming for rapid credit assignment in the context of multi-step off-policy RL. Our method does not involve multi-step updates; instead, it applies value function compositions. Sample efficiency is also related to the problem of exploration in RL [1, 36, 42, 44]

In goal-reaching tasks [2, 15, 32, 33], goals are typically distant and widely distributed. Thus, efficient information propagation is crucial to perform well in such environments—a challenge known as the long-horizons problem [10, 12]. Park et al. [27], like us, addresses the long-horizons problem by creating a flat policy from a single value function. However, it does not introduce a hierarchical inductive bias in the value function to accelerate value propagation.

Our method is closely related to hierarchical RL[6, 28, 34, 35, 37, 48]. Hierarchical RL generates subgoals to aid in problem-solving—after identifying a subgoal, it focuses on reaching the subgoal, thereby forming a hierarchical policy. In contrast, our method employs a flat policy, utilizing subgoals solely for more efficient information propagation through better problem decomposition. In this

context the closest related work to us are Dhiman et al. [8], Kaelbling [15]. Our formulation, however, allows to generalize the algorithm into deep RL case.

Our algorithm can also be easily integrated with hierarchical methods, serving as a subgoal discovery mechanism [49].

B Method

The Bellman equation decomposes the value function into two parts: the reward obtained at the immediate next step, and the expected discounted reward gathered over all subsequent steps ($\gamma V(s')$). This intuitive formulation makes learning possible even in scenarios with infinite episodes. However, methods based on the direct application of the Bellman optimality equation necessitate accurate estimates for neighboring states to obtain a reliable estimate for the updated states. Therefore, information propagates linearly in terms of loop iterations or target network updates.

A viable strategy to increase the speed of information propagation is employing multi-step updates that roll out the Bellman equation beyond a single step and use nearer estimates for the update. However, multi-step methods, when applied with off-policy algorithms [30, 41], suffer from high-variance [5]. Methods, such as Harutyunyan et al. [13], Munos et al. [24], have been proposed to reduce the variance of multi-step methods. We propose an alternative approach that uses the philosophy of multi-step methods but, by bootstrapping, avoids high variance related to the sampled rollout.

In the context of goal-conditioned RL, the agent knows the goal it is trying to reach from the start of the episode. We leverage this information to derive a method that significantly accelerates information propagation, achieved by revisiting the Bellman equation. Our *Compositional Optimality Equation* also partitions the problem into two subproblems that are connected by an interim state that we call a subgoal. Unlike the Bellman optimality equation, our intermediate states (subgoals) do not have to be adjacent to the state being updated. Therefore, they can segment the trajectory into more equidistant parts, thereby enabling exponentially faster value propagation.

B.1 Compositional Optimality Equation

We focus on Goal-conditioned RL. We make the assumption that the state space is the same as the goal space, denoted as $\mathcal{S}_S = \mathcal{S}_G$. Recall the extended state space $S = \mathcal{S}_S \times \mathcal{S}_G$. Assume that the underlying MDP is deterministic. In other words, $\mathcal{T}$ is not a distribution, but a function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Let Π_{opt} represent the set of all deterministic policies that are optimal, such that $\forall_{s,g \in S, \pi \in \Pi_{opt}} V^\pi(s,g) = V^*(s,g)$. Moreover, Π_{opt} is not empty as a global optimal policy always exists. Let $T_\pi(s,g)$ denote a trajectory (a set of visited states) of π when run on state s and goal g.

A state $subg$ is called a subgoal of (s,g) if there exists a deterministic optimal policy that visits $subg$ on the way from s to g. In other words, $subg$ is a subgoal from s to g if it is on some optimal path from s to g. Formally:

$$subg \text{ is a subgoal from s to g } \iff subg \in \bigcup_{\pi \in \Pi_{opt}} T_\pi(s,g)$$

. We define SUBG(s,g) = $\bigcup_{\pi \in \Pi_{opt}} T_\pi(s,g)$ as the set of subgoals from s to g.

An alternative way to define the same set $SUBG$ involves value functions instead of trajectories. Specifically:

$$SUBG_V(s,g) = \{s' \in \mathcal{S}_S : V^*(s,g) = V^*(s,s')V^*(s',g)\}$$

.

Lemma 1: Equivalence of definitions:

$$SUBG(s,g) = SUBG_V(s,g)$$

.

Proof First, we demonstrate why, in Goal-conditioned deterministic environments, we can examine distance functions instead of value functions.

Let $d_\pi(s,g)$ be a random variable representing a distance from s to g under policy π. In other words, if $T_\pi(s,g)$ is a sample rollout of π from s to g, then $d_\pi(s,g) = |T_\pi(s,g)| - 1$. We also define $d(s,g) := \min_\pi d_\pi(s,g)$ as the optimal distance from s to g.

from the Bellman expectation equation, we derive $V^\pi(s,g) = \mathbf{E}[\gamma^{d_\pi(s,g)}]$. Since we operate on deterministic environments and focus on deterministic policies, $V^\pi(s,g) = \gamma^{d_\pi(s,g)}$. Thus, $V^*(s,g) = \max_\pi V^\pi(s,g) = \gamma^{\min_\pi d_\pi(s,g)} = \gamma^{d(s,g)}$.

To show that $SUBG(s,g) = SUBG_V(s,g)$ we must demonstrate:

$$V^*(s,g) = V^*(s,subg)V^*(subg,g) \iff \exists_{\pi \in \Pi_{opt}} subg \in T_\pi(s,g).$$

Starting with the left-to-right implication. Let π be a deterministic, optimal policy. Then from the assumption we have that $V^\pi(s,g) = V^\pi(s,subg)V^\pi(subg,g)$, which is equivalent to $d^\pi(s,g) = d^\pi(s,subg) + d^\pi(subg,g)$. Therefore, $subg$ must belong to $T_\pi(s,g)$, as otherwise $d^\pi(s,g) \leq d^\pi(s,subg)$ contradicts the sum of distances. Thus, π is the policy from Π_{opt} such that $subg \in T_\pi(s,g)$

The proof in the opposite direction is also straightforward. Let π be a policy that satisfies the right-hand side assumptions, with $\pi \in \Pi_{opt}$ such that $subg \in T_\pi(s,g)$. Note that $T_\pi(s,subg) \bigcup T_\pi(subg,g)$ does not necessarily have to be equal to $T_\pi(s,g)$. However, because π is optimal everywhere, we must have $d_\pi(s,g) = d_\pi(s,subg) + d_\pi(subg,g)$ for $subg$ is in $T_\pi(s,g)$. Otherwise, π would not be optimal at $(s,subg)$ or at $(subg,g)$. Therefore, $V^*(s,g) = V^\pi(s,g) = \gamma^{d_\pi(s,subg)+d_\pi(subg,g)} = V^\pi(s,subg)V^\pi(subg,g) = V^*(s,subg)V^*(subg,g)$.

Triangle inequality The decomposition of the value function into the product of values at any state (not necessarily optimal) satisfies the triangle inequality.

$$\forall_{s'} V^*(s,g) \geq V^*(s,s')V^*(s',g). \tag{1}$$

This emerges from the triangle inequality of distances expressed in the exponential form: $V^*(s,g) = \gamma^{d(s,g)} \geq \gamma^{d(s,s')+d(s',g)} = \gamma^{d(s,s')}\gamma^{d(s',g)} = V^*(s,s')V^*(s',g)$.

Consequently, we obtain an alternative optimality equation.

Compositional Optimality Equation

$V^*(s,g) = V^*(s,subg)V^*(subg,g)$ for any subg $\in SUBG(s,g) = \max_{subg \in S} V^*(s,subg)V^*(subg,g)$

We base our algorithm for subgoal discovery and an alternative update rule on this equation.

B.1.1 Proof of convergence of Compositional Value Iteration

Let $V^k(s,g)$ be the estimate of $V^*(s,g)$ after k steps. We will show by induction that after k iterations of the algorithm, all pairs of states that are closer to each other than 2^k have already reached optimal value function as their $V^k(s,g)$ estimates. We will also show that all estimates are smaller than or equal to the optimal value function. In other words, after k iterations we have two conditions satisfied: 1) $\forall_{(s,g):d(s,g)\leq 2^k} : V^k(s,g) = V^*(s,g)$ and 2) $\forall_{(s,g)} V^k(s,g) \leq V^*(s,g)$.

After 0 iterations (upon the first time visiting line 7), we have $\forall_{(s,g):d(s,g)=1} V^0(s,g) = \gamma$ and $\forall_{(s,g):d(s,g)=0}$, also $\forall_{(s,g):s\neq g} V^0(s,g) = 0 \leq V^*(s,g), \forall_s V^0(s,s) = 1 \leq V^*(s,s)$. Thus, the base of the induction is satisfied.

Now for the k'th step, assume that our assumption holds for $k-1$. First, $\forall_{s,g} V^{k-1}(s,g) \leq V^*(s,g)$, so as a consequence of triangle inequality 1 after the update V will also satisfy $V^k(s,g) \leq V^*(s,g)$.

Let us consider $(s,g) : d(s,g) \leq 2^k$. Then, let us choose a subgoal from some optimal trajectory that is in the middle, such that $d(s,subg) \leq 2^{k-1}$ and $d(subg,g) \leq 2^{k-1}$. From our assumption $V^{k-1}(s,subg) = V^*(s,subg), V^{k-1}(subg,g) = V^*(subg,g)$. Therefore if $subg$ is selected for an update, we will have $V^{k-1}(s,subg)V^{k-1}(subg,g) = V^*(s,subg)V^*(subg,g) = V^*(s,g)$.

	Mean	Min	Max
COE-DDPG	19.125	11	35
DDPG	32.187	24	38
Relative gain	**40.6%**	**54.2%**	7.9%

Table 2: Comparison of sample efficiency of COE-DDPG with DDPG on the continuous gridworld. The table is a summary of 32 runs per algorithm. For each run, we calculate the first epoch in which all episodes were solved. Relative gain is the percentage difference between COE-DDPG and DDPG. Both average-case and best-case are significantly better for COE-DDPG than DDPG.

However, as we are maximizing over all possible subgoal states, it follows that $V^k(s,g) \geq V^*(s,g)$, since we have to choose a state that yields a product at least as large as $subg$.

In summary, $V^k(s,g) \geq V^*(s,g)$, but also $V^k(s,g) \leq V^*(s,g)$. So $V^k(s,g) = V^*(s,g)$ which proves the inductive step.

C Experimental details

C.1 Gridworld

We evaluate our COE Value Iteration (COE-VI) on a two-dimensional gridworld environment composed of 50×50 tiles and compare it to standard Value Iteration (VI). For each generated board, each position is designated as a wall with a probability of 0.3, and as a floor tile otherwise. Subsequently, we sample the start state and the goal state uniformly from all floor positions. Next, we check algorithmically whether the goal is reachable from the start state. If not, then we repeat the sampling procedure. Otherwise the RL agent receives it as a task. Our results are consistent with theoretical derivations and confirm that COE Value Iteration is exponentially faster in terms of the number of updates. The results are presented in Table 1. Notice that COE-VI requires fewer updates to solve a problem with a state-to-goal distance of 10000 than VI requires to solve one of distance 25. The number of steps necessary to learn to reach the goal at a distance d with COE-VI is exactly $\lceil log_2(d) \rceil$, whereas for standard VI, it is d.

C.2 Continuous gridworld

To evaluate the sample efficiency of Deep RL methods derived from COE, we compare COE-DDPG with DDPG in a continuous gridworld environment to determine the speed at which each solves the environment. The continuous gridworld used for the comparison has a state space of $[-20, 20]^2$, and an action space of $[-1, 1]^2$. In each episode, the start and the goal states are uniformly sampled from the state space. A small centralized Gaussian noise with a standard deviation of 0.1 is added to the action at each step. The goal is considered achieved if the distance between the player's position and the goal position is less than 0.5.

For our agent, we use COE-DDPG (3).We compare the results of COE-DDPG to the standard DDPG [18]. All models—policy, state-action value functions, and generator—are dense neural networks. Both the policy and generator contain three hidden layers, whereas the value function has four. All models have the same hidden dimension of 512. The generator and policy employ the tanh activation function with suitable scaling. For the value function, the sigmoid activation function is used. The learning rates, denoted as lr_q, lr_π, lr_G are consistent and set to 10^{-5}. All models are trained with the Adam optimizer [16].

To evaluate the performance, we execute 32 experiments for each algorithm and study the convergence speed of both. Training starts with 50,000 explorative steps. During this phase, the agent interacts with the environment using a randomly initialized policy without improvement, solely collecting data for the experience replay buffer, which has a capacity of $50,000$. Subsequently, the initial epoch of training begins. This epoch comprises 3000 network update steps and 5000 steps of interaction with the environment, accumulating data for the experience replay [19]. Each episode is limited to 50 steps, after which it is truncated. We measure the number of epochs necessary to solve the problem. Specifically, we record the initial epoch that achieves a 100% success rate across its episodes.

The results are shown in Table 2. On average, COE-DDPG proves to be 40.6% faster than its conventional counterpart, with the fastest run being 54.2% faster than the original. In the least favorable scenarios, the advantage is smaller, although closer inspection reveals that these least favorable runs are outliers for COE-DDPG. A more in-depth view of the distribution of runs can be seen in Figure 1. This Figure indicates that the distribution associated with COE-DDPG is distinctly skewed to the left.

While we observe significant improvements from using COE-DDPG over using standard DDPG in this setting, they do not match the exponential speedup witnessed in the discrete case. This disparity results from three factors. First, due to their smoothness, neural networks propagate value updates into nearby states, mitigating the exponential speedup effect. Next, in a discrete environment, the update encompasses all states. In contrast, in a continuous setting, the variance introduced by the generator increases the variance of the entire method. Finally, the discrete algorithm assumes that the values are initialized at 0. This assumption is problematic in function approximation as it may lead to either vanishing gradients or other complex formulations.

D COE-based Deep RL algorithms

Algorithm 2 COE-DQN

Input: Q-function parameters θ_q, generator parameters θ_g, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{q,t} \leftarrow \theta_q, \theta_{g,t} \leftarrow \theta_g$
repeat
  Observe state s and goal g and select action $a = \max_a Q_{\theta_{q,t}}(s, a, g)$
  Execute a in the environment
  Observe next state s', reward r, goal g, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', g, d) in replay buffer $\mathcal{D}$
  if s' is terminal, reset environment state.
  if it's time to update **then**
    if first time updated **then**
      Initialize generator with Bellman Generator Initialization (See **??**)
    end if
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', g, d)\}$ from $\mathcal{D}$
      Generate a subgoal for the update $subg = G_{\theta_g,t}(s, a)$
      Compute targets

$$y(s, a, subg, g) = Q_{\theta_{q,t}}(s, a, subg) \max_{a'} Q_{\theta_{q,t}}(subg, a', g)$$

      Update Q-function by one step of gradient descent using

$$\nabla_{\theta_q} \frac{1}{|B|} \sum_{s,a,r,s',g,d) \in B} (Q_{\theta_q}(s, a, g) - y(s, a, subg, g))$$

      Update the generator by one step of gradient ascent using

$$\nabla_{\theta_g} \frac{1}{|B|} \sum_{(s,g) \in B} (1-d) Q_{\theta_{q,t}}(s', a, subg) \max_{a'} Q_{\theta_{q,t}}(subg, a', g) \text{ where } subg = G_{\theta_g}(s, a)$$

      Update the Q-function by one step of gradient descent on next states from the batch

$$\nabla_{\theta_q} \frac{1}{|B|} \sum_{(s,a,s') \in B} (Q_{\theta_q}(s, a, s') - \gamma)$$

      Update target networks
    end for
  end if
until convergence

Algorithm 3 COE-DDPG

Input: initial policy parameters θ_p, Q-function parameters θ_q, generator parameters θ_g, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{p,t} \leftarrow \theta_p, \theta_{q,t} \leftarrow \theta_q, \theta_{g,t} \leftarrow \theta_g$
repeat
  Observe state s and goal g and select action $a = clip(\mu_{\theta_p}(s,g) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, goal g, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', g, d) in replay buffer $\mathcal{D}$
  if s' is terminal, reset environment state.
  if it's time to update **then**
    if first time updated **then**
      Initialize generator with Bellman Generator Initialization (See **??**)
    end if
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', g, d)\}$ from $\mathcal{D}$
      Generate a subgoal for the update $subg = G_{\theta_g,t}(s, a)$
      Compute targets

$$y(s, a, subg, g) = Q_{\theta_{q,t}}(s, a, subg) Q_{\theta_{q,t}}(subg, \mu_{\theta_{p,t}}(subg, g), g)$$

      Update Q-function by one step of gradient descent using

$$\nabla_{\theta_q} \frac{1}{|B|} \sum_{s,a,r,s',g,d) \in B} (Q_{\theta_q}(s, a, g) - y(s, a, subg, g))$$

      Update policy by one step of gradient ascent using

$$\nabla_{\theta_p} \frac{1}{|B|} \sum_{(s,g) \in B} Q_{\theta_q}(s, \mu_{\theta_p}, g)$$

      Update the generator by one step of gradient ascent using

$$\nabla_{\theta_g} \frac{1}{|B|} \sum_{(s,g) \in B} (1-d) Q_{\theta_{q,t}}(s', a, subg) Q_{\theta_{q,t}}(subg, \mu_{\theta_{p,t}}(subg, g), g) \text{ where } subg = G_{\theta_g}(s, a)$$

      Update the Q-function by one step of gradient descent on next states from the batch

$$\nabla_{\theta_q} \frac{1}{|B|} \sum_{(s,a,s') \in B} (Q_{\theta_q}(s, a, s') - \gamma)$$

      Update target networks
    end for
  end if
until convergence