Time is of the Essence: Why Decision-Time Planning Costs Matter

Kevin A. Wang
kevinwang@kevinwang.us
Brown University

Jerry Xia
Brown University

Stephen Chung
University of Cambridge

Jennifer Wang
Brown University

Francisco Piedrahita Velez
Brown University

Hung-Jen Wang
Brown University

Amy Greenwald
Brown University

Abstract

The goal of this work is to build agents that use resources efficiently during decision-time planning. Such agents should learn to dynamically spend more compute at difficult, critical decision points, and less otherwise. To this end, we propose timed MDPs and timed policies, which augment MDPs and policies to explicitly factor in the cost of time and compute usage. By extending MDPs in this way, agents can learn to trade off the cost of planning against potentially higher rewards.

To make our point concretely, we modify an existing algorithm, Thinker, to use a variable amount of compute to make each decision. We then train it to maximize a reward that includes a penalty for using more compute that depends on the context. Our modified algorithm, Dynamic Thinker, learns to use compute more efficiently than Thinker and AlphaZero. More specifically, it reaches similar performance levels using fewer planning steps in experiments on a simple knapsack problem.

1 Introduction

Modern RL agents use neural networks whose parameters are learned during training. In typical deep RL agents (Mnih et al., 2013; Schulman et al., 2017), the agent chooses an action by simply querying a neural network, which takes a small, constant amount of time and compute resources. Alternatively, an agent can spend a non-trivial amount of time and compute to decide on an action, a phase called **decision-time planning (DTP)** (or more colloquially, **search**). A prominent example of this is the AlphaZero family of agents (Silver et al., 2018).

DTP has indeed proven beneficial in domains like chess, go (Silver et al., 2016), and poker (Moravčík et al., 2017; Brown & Sandholm, 2019). When agents can access state-specific information, additional compute spent planning can yield local policy improvements (Sutton & Barto, 2018; Bertsekas, 2022).

With the advent of deep learning, more and more aspects of DTP are being controlled by learned parameters, rather than hand-crafted heuristics (Schrittwieser et al., 2020; Brown et al., 2020; Schmid et al., 2023; Sokota et al., 2021; Guez et al., 2019; Lehnert et al., 2024; Guez et al., 2018; Sychrovský et al., 2024; Anthony, 2021). Still, AlphaZero's search strategy uses a hand-crafted heuristic (i.e., a precise formula) to trade-off exploration against exploitation, before expanding the node visited most often during the search phase, another hard-coded decision (Grill et al., 2020). Going beyond some of AlphaZero's hand-crafted heuristics, the Thinker algorithm (Chung et al., 2023) is more flexible in its choice of where to search, and in its choice of which action to expand upon completing its search.

In other words, Thinker seeks to learn a policy to guide not only its "real" planning (i.e., its reward-accruing actions), but its decision-time planning actions as well. While Thinker's abilities are impressive, and undoubtedly a step in the right direction, Thinker is incapable of explicitly reasoning

about the tradeoff between the better performance that can be obtained by investing resources in decision-time planning vs. its higher cost, because Thinker does not reason about DTP costs.

It seems to us uncontroversial to claim that DTP costs should not be ignored: agents that use constrained resources more efficiently than others are preferable. In some applications, such as timed chess and go, the resources available for decision-time planning are explicitly constrained. In others, users may simply prefer not to interact with agents that make them wait.

To address this limitation, we develop **timed MDPs**—and correspondingly, **timed policies**—in this work, which augment a given MDP by tracking time in the state. Rewards likewise reflect DTP costs, by combining them with the rewards in the underlying MDP. By learning an approximately optimal timed policy in a timed MDP, an agent can automatically trade off between DTP costs and rewards.

N.B. Examples of compute resources include wall time, CPU cycles, and energy consumption. For simplicity, in this paper we refer to all resources used while planning as "time."

The standard Thinker algorithm searches for a fixed number of steps at each state. To demonstrate the utility of timed MDPs, we modify Thinker so that it can plan for a flexible number of steps at each state, which we call Dynamic Thinker, and use it to solve a timed MDP. The trained agent learns how much to search based on the context. The timed MDP framework enables our agent to trade off DTP costs against potentially greater rewards. In experiments on a simple knapsack problem, we show that Dynamic Thinker attains performance levels similar to AlphaZero and Thinker, incurring smaller DTP costs (i.e., using fewer planning steps).

2 Modeling Time

Suppose Alice is designing an agent to act in a sequential environment. First she models the environment (e.g., as an MDP). Then she picks a corresponding hypothesis class for her agents (e.g., policy neural networks with n layers of m neurons each). Finally, she selects an agent from the hypothesis the class that maximizes rewards (e.g., through reinforcement learning). As Sutton & Barto (2018) note, "the success of a reinforcement learning application strongly depends on how well the reward signal frames the goal of the application designer and how well the signal assesses progress in reaching that goal." Thus, Alice's goal is to model the environment—and the reward function, in particular—accurately enough so that the choice agent is one that behaves as she intended.

In this section, we first present the typical formulation of MDPs and policies. Although others have built MDPs that incorporate decision-time planning actions (e.g., Thinker (Chung et al., 2023)), they do not typically account for resource usage during decision-time planning. To mitigate this deficiency, we propose **timed MDPs** and **timed policies**, which extend MDPs and policies to model DTP costs, thereby enabling agents to reason about the tradeoffs between the benefits of resource use and its costs.

2.1 Traditional MDPs and Policies

When building an RL agent, we typically model the environment as a **Markov decision process (MDP)** or a variant thereof,[1] and we model the agent as a **policy**.

We write $\triangle(\mathcal{X})$ to denote the set of probability distributions over an arbitrary (finite) set $\mathcal{X}$.

An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $P : \mathcal{S} \times \mathcal{A} \to \triangle(\mathcal{S})$ is a probabilistic transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1]$ is a discount factor, and μ is an distribution over initial states.

[1] Other variants of the MDP include the partially observable MDP (POMDP), the Decentralized POMDP (Dec-POMDP), the Markov game (Shapley, 1953), and the factored-observation stochastic game (Kovařík et al., 2021).

An MDP starts at time step $t = 0$, with the agent in a state s_0 randomly drawn from μ. It then samples an action a_0 from $\pi(s_0)$. The environment then draws a new state s_1 from $P(s_0, a_0)$, and the agent receives reward $r_1 = R(s_0, a_0)$. This process then repeats.

A **policy** $\pi : \mathcal{S} \to \triangle(\mathcal{A})$ is a function that maps each state to a distribution over actions. Every policy π in an MDP induces an expected cumulative discounted return $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi\right]$. An optimal policy is one that maximizes this return.

A typical policy does not model decision-making time of the agent. Hence, we refer to such policies as **untimed policies** (to distinguish them from their timed counterparts we define in Section 2.2). An untimed policy is an excellent model for the types of agents that are typically studied in RL, namely "fast" agents that do not use significant amounts of time or compute to make decisions. For instance, RL agents may simply query a policy table, a Q-value table, or their neural network counterparts. Then, they return either the result of the query, or the result of extremely simple computation on the results, such as taking the argmax. This process takes a small, constant amount of time.

Now, consider a hypothesis class of agents, some of which use more compute than others to make decisions. We call such hypothesis classes **time-varying**. Two time-varying agents π and ρ may implement the same untimed policy; that is, they may choose the same actions at all states. However, they may also reach these decisions using different amounts of compute. If ρ takes 1 year of computation to reach each decision, while π only takes 1 second, π is preferable.

When building a search agent, such as AlphaZero for chess, the hypothesis class is typically time-varying: the amount of search used at each state is a parameter that can be tuned. To date, engineers have largely controlled this parameter by designing hand-crafted heuristics. However, as search algorithms are becoming increasingly general, and governed more and more by learned parameters, it seems reasonable to also incorporate parameters that dictate search time, as well as cost functions that evaluate resource usage. Agents would then learn to reason about the benefits of resource use vs. its costs.

To address this issue, we propose a way to extend MDPs to explicitly model time and other computational resources.

2.2 Timed MDPs and Timed Policies

We denote by $\mathcal{C} \subseteq \mathbb{R}$ the range of possibilities of compute resource usage. We mostly refer to a generic compute resource which we call "time" for simplicity, but this sole resource can be a conglomeration of multiple resources (e.g., time and compute), and usage can be measured in other units. For example, to model wall-time in seconds, $\mathcal{C}$ might be the positive reals $\mathbb{R}^+$, while to model the number of operations under some model of computation, $\mathcal{C}$ might be $\mathbb{N}$.

Given an MDP $(\mathcal{S}', \mathcal{A}', \mathcal{P}', \mathcal{R}', \gamma, \mu)$, we construct a **timed MDP** $(\mathcal{C}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, u, \gamma, \mu)$ as follows:

- $\mathcal{C} \subseteq \mathbb{R}$ is the range of possibilities of compute resource usage (i.e., times).
- $\mathcal{S} = \mathcal{S}' \times \mathcal{C}$ is a set of states
- $\mathcal{A} = \mathcal{A}' \times \mathcal{C}$ is a set of actions
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \triangle(\mathcal{S})$ is the **timed transition function**
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the **timed reward function**, which is defined based on $\mathcal{R}'$ and a timed cost function $u : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$. A straightforward example is $\mathcal{R}((s', C), (a', c)) = \mathcal{R}'(s', a') + u(C, c)$, for all $s' \in \mathcal{S}'$, $a' \in \mathcal{A}'$, and times $C \in \mathcal{C}$.
- $\gamma \in [0, 1]$ is the discount factor
- μ is the distribution of initial states

A timed MDP augments a given MDP by adding to the action space a variable that represents an amount of planning "time" (i.e., compute resource usage) to choose an action. It further adds this time variable to the state space, in order to track the total time used for decision-time planning.

As in a(n untimed) MDP, an initial state s_0 is drawn from μ, and the total time C_0 is initialized to 0. The agent receives s_0 as input, based on which it chooses action a_0, which costs c_0 units of time. The agent then receives $\mathcal{R}((s_0, C_0), (a_0, c_0))$ reward, the state transitions to s_1 drawn from $\mathcal{P}((s_0, C_0), (a_0, c_0))$, and the total time becomes $C_1 = C_0 + c_0$.

By what criteria should one define u for a specific application?

For some tasks, u is given: e.g., in the timed games of chess or go, or in combinatorial auctions with strict time limits on placing bids.

For other tasks, u should be chosen to reflect the intent of the agent's designer. For example, a chatbot user may not want to wait too long before receiving a reply to her queries. To minimize user frustration in such situations, it would be best to train a chatbot on user preference data which takes the time used to generate each response into account.

Some environments are harder: if a team of researchers working at a company trains an agent with a time-varying policy to find drugs, how are they to balance solution quality with search time? Should they take into account the monetary cost of compute? In such a scenario, it may be impossible to quantify u. But note that if the search time were controlled by hand-picked hyperparameters rather than learned parameters, then some decision would be taken for the values of the hyperparameters. Therefore, the goal of modelling u is not ill-posed.

Finally, an environment may not directly model a real-world task; it might instead be used as a tool, for example, to benchmark RL algorithms. In this case, if the environment is meant to model some aspect of real-world tasks, then the choice of u should correspond to that aspect.

Possible classes of timed cost function u include:

- Linear cost function: $u(C, c) = \alpha c$, for some constant $\alpha < 0$. This cost function models a constant cost per unit of time.
- Soft time budget per decision: $u(C, c) = \min(0, \beta - c)$, for some constant $\beta > 0$. This cost function can be used to model scenarios where the amount of time used does not matter until it exceeds a certain amount. For example, a chatbot that outputs text word-by-word for a human to read should not take too long to generate the next word, but speed improvements beyond the speed at which the human can read have no marginal benefit.
- Hard time budget per decision: $u(C, c) = \begin{cases} 0 & c < \beta \\ -\infty & c \geq \beta \end{cases}$, for some constant $\beta > 0$. This cost function allows us to codify the previously unformalized assumption in MDPs that agents take a small, constant amount of time to make their decisions.
- Hard time budget per episode: $u(C, c) = \begin{cases} 0 & C < \beta \\ -\infty & C \geq \beta \end{cases}$, for some constant $\beta > 0$. Games like chess and go with time controls can be modeled in this way.

3 Dynamic Thinker

3.1 Thinker

Thinker (Chung et al., 2023) is a tree search algorithm that is heavily controlled by learned parameters. Because of its ability to learn parameters that are hardcoded in other tree search algorithms, it can be thought of as a generalized version of other algorithms like AlphaZero.

In Thinker, the search tree is considered external to the agent. An MDP is transformed into an augmented MDP, in which the agent can not only take actions in the original MDP, but it can also

take actions to explore nodes in the search tree, which we call **planning actions**. Both types of actions are chosen conditionally on the state of the current search tree.

We refer to the agent in the augmented MDP as the **internal agent**. We refer to the induced agent in the original MDP as the **holistic agent**.

The internal agent itself is "fast", yet by acting in its search-tree environment, it performs planning. That is, the internal agent uses a small, fixed amount of compute resources per decision, yet it spends a large amount of compute between real actions in the real environment (i.e., by taking multiple planning actions before taking a real action).

3.2 Dynamic Thinker

Thinker uses a fixed number of planning actions before it takes a real action. The number of planning actions, denoted k, is a hyperparameter.

We modify Thinker so that it has an extra action: "finish search." If this action is taken, then the agent's next action is a real action. We call this modified algorithm **Dynamic Thinker**.

Since Dynamic Thinker can use a variable number of planning actions, the parameterization of the holistic agent is a time-varying hypothesis class.

Further, the optimization of the *timed policy* of the holistic agent can be performed through deep reinforcement learning.

4 Experiments

So far, we have argued that objective functions for typical, untimed MDP are impoverished, since they disregard the computational costs of decision-time planning, and that augmenting an MDP with timed rewards can better model the environment. In this section, we that demonstrate that it is actually possible to take advantage of timed MDPs. Specifically, we report on experiments in which we trained a search agent to maximize timed rewards. The result is an agent that dynamically chooses a variable amount of compute to spend per decision.

In these experiments, we choose the number of internal agent actions (both search tree traversals and real actions) as our unit of compute ("time"). We model planning cost as a constant negative reward for each traversal and each real action.

We use a simple environment based on the standard knapsack problem. In the problem, you are given a set of N items and their respective weights $w_i \in \mathbb{N}$, as well as a knapsack that can carry up to M pounds, and your goal is to pick some subset of the items with maximum total weight, without exceeding the knapsack's capacity. In our experiments, we use M=10 and N=5.

In our experiments, the agent starts with an empty knapsack, and at each state, can pick one item to add to the knapsack. An experiment terminates when no further items can be added to the knapsack. The agent receives a reward of w_i when it adds item i to the knapsack. If the selected item is illegal (either it has already been used, or it would exceed the capacity of the knapsack), then the agent receives -0.4 reward and the episode continues. These rewards are normalized by dividing by the maximum achievable total weight for the instance, so that the optimal policy will receive a cumulative reward of 1.

Each time the internal agent takes an action—that is, it traverses a node in its search tree, or it decides on a real action—it incurs a -0.001 reward.

More formally, let $\mathcal{M}$ be the set of items and let w_i be the reward for legally adding item i to the knapsack. We model the knapsack problem as a timed MDP:

- The set of compute resource usage $\mathcal{C} = \mathbb{N}$ is the number of planning steps taken.

- The state space $\mathcal{S} = 2^{\mathcal{M}}$ is the set of all possible sets of items in the knapsack.
- The action space $\mathcal{A} = \mathcal{M}$ includes an action for adding each item to the knapsack.
- The timed state space $\mathcal{S} \times \mathcal{C}$ tracks the number of planning steps taken up to the current state. We use the tuple (S, C) to represent a timed state, where S is the set of items currently in the knapsack and C is the number of planning steps used so far.
- The timed action space $\mathcal{A} \times \mathcal{C}$ includes the number of planning steps used to select the next item $i \in \mathcal{A}$ to put in the knapsack. We use the tuple (i, c) to represent a timed action where i is the selected item and c is the number of planning steps used to compute the action.
- At a timed state (S, C) and a timed action (i, c), the timed transition function places item i into the knapsack and adds the number of planning steps taken to the total, if the item can be legally added: $(S \cup \{i\}, C + c)$. Otherwise, the total planning step counter is incremented and the knapsack is unchanged: $(S, C + c)$.
- The timed reward function returns $w_i + \alpha c$ if item i can be legally placed into the knapsack, or αc if i is illegal. The hyperparameter α is a small fixed reward penalty given for each planning step.

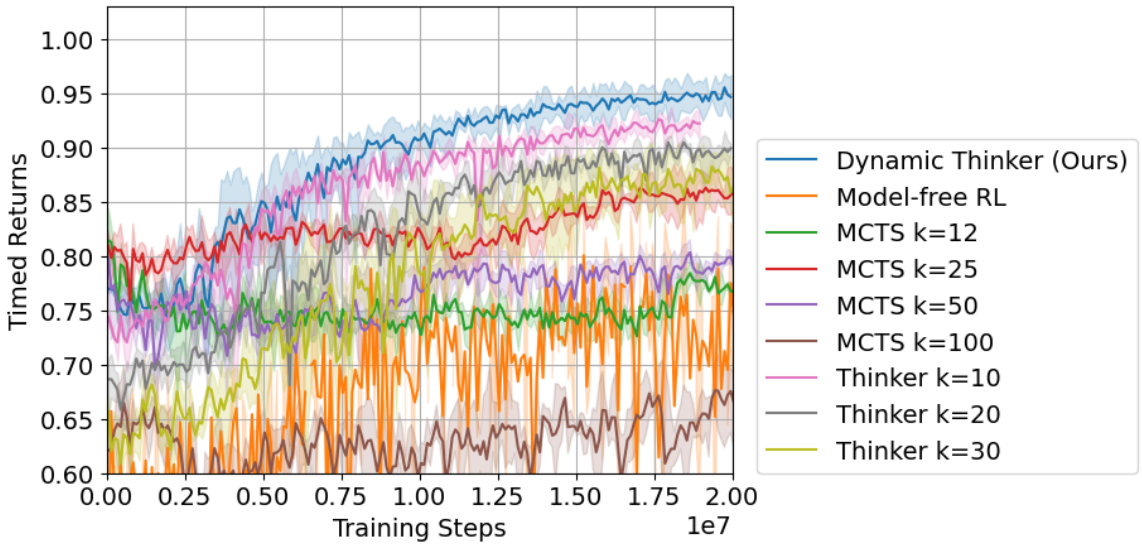


Figure 1: Main results: timed returns during training. The y-axis shows the cumulative timed rewards ($\mathcal{R}$)

We do not sort the items in the representation, and illegal actions are penalized, not masked. These difficulties mean that learning an optimal policy for this environment is not trivial, despite its small size. Learned tree search can be very useful: an agent can easily learn to use even 1-step lookahead to avoid illegal actions. However, doing more search is not always useful: for example, if search finds a set of items that results in the maximum capacity (10), then no further search is required.

We compare Dynamic Thinker with the following baselines:

1. Unmodified Thinker with various amounts of fixed search per decision ($k \in \{10, 20, 30\}$)
2. AlphaZero-style Monte Carlo tree search with various amounts of fixed search per decision ($k \in \{12, 25, 50, 100\}$)

3. Model-free actor-critic agent ($k = 1$) (this is the same actor-critic algorithm as the internal agent)

Figure 1 shows the cumulative timed rewards accumulated by each of the methods over the course of training. The x-axis counts the number of times the internal agent took an action in the environment or took a planning action. After 2e7 such steps, Dynamic Thinker achieves returns of around 0.95, more than any other method.

Some other methods are able to achieve similar or higher returns as measured by the original, time-independent reward function, shown in Figure 2. Results with extended x-axes appear in Appendix C.2

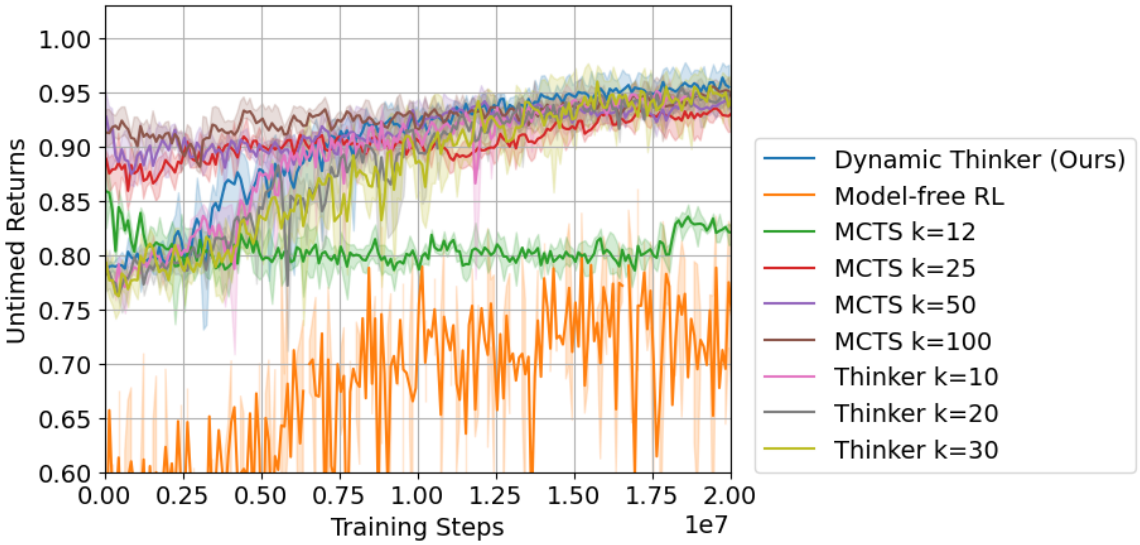


Figure 2: Time-independent returns during training. The y-axis shows the cumulative untimed rewards (R)

5 Conclusion

Decision-time planning algorithms with learned parameters are increasingly popular. In this paper, we argued that introducing time-varying hypothesis classes into decision-time planning algorithms will allow us to automatically learn agents that make the preferred trade-off between better performance and greater computational cost. However, unless we use a timed reward function, there will likely be a mismatch between the agents that we get and the agents that we want. For our models to accommodate such a reward function, we propose that we should model agents as timed policies, and model environments as timed MDPs.

Additionally, we created a toy environment, and modified an existing tree search algorithm, Thinker, to create a dynamic variant that learns a timed policy in a timed MDP. We trained Dynamic Thinker, and observed that it achieved higher timed returns than any other method , which demonstrates that agents that operate in timed MDPs can learn to manage resources efficiently, ameliorating the need for hand-crafted heuristics and hard-coded hyperparameters.

In the future, we plan to develop timed MDPs as benchmarks for a variety of real-world tasks to foster additional study of DTP agents that learn to manage resources efficiently.

Acknowledgments

We thank David Wu for conversations which helped to inspire and guide this line of research. We thank George Konidaris, Michal Šustr, and Adrian Orenstein for helpful conversations. We thank the reviewer for their helpful feedback.

References

Thomas William Anthony. *Expert iteration*. Doctoral, UCL (University College London), March 2021. URL https://discovery.ucl.ac.uk/id/eprint/10123580/. Conference Name: UCL (University College London) Meeting Name: UCL (University College London) Publication Title: Doctoral thesis, UCL (University College London).

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to Ponder, September 2021. URL http://arxiv.org/abs/2107.05407. arXiv:2107.05407 [cs].

Dimitri P. Bertsekas. *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*. Athena Scientific, March 2022. ISBN 978-1-886529-17-5.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, August 2019. doi: 10.1126/science.aay2400. URL https://www.science.org/doi/full/10.1126/science.aay2400. Publisher: American Association for the Advancement of Science.

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games, November 2020. URL http://arxiv.org/abs/2007.13544. arXiv:2007.13544 [cs].

Stephen Chung, Ivan Anokhin, and David Krueger. Thinker: Learning to Plan and Act. *Advances in Neural Information Processing Systems*, 36:22896–22933, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/4761fab863f0900d90cf601fce6d5155-Abstract-Conference.html.

Jean-Bastien Grill, Florent Altché, Yunhao Tang, Thomas Hubert, Michal Valko, Ioannis Antonoglou, and Remi Munos. Monte-Carlo Tree Search as Regularized Policy Optimization. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3769–3778. PMLR, November 2020. URL https://proceedings.mlr.press/v119/grill20a.html. ISSN: 2640-3498.

Arthur Guez, Théophane Weber, Ioannis Antonoglou, Karen Simonyan, Oriol Vinyals, Daan Wierstra, Rémi Munos, and David Silver. Learning to Search with MCTSnets, July 2018. URL http://arxiv.org/abs/1802.04697. arXiv:1802.04697 [cs, stat].

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sebastien Racaniere, Theophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, Greg Wayne, David Silver, and Timothy Lillicrap. An Investigation of Model-Free Planning. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2464–2473. PMLR, May 2019. URL https://proceedings.mlr.press/v97/guez19a.html. ISSN: 2640-3498.

E. A. Hansen and R. Zhou. Anytime Heuristic Search. *Journal of Artificial Intelligence Research*, 28:267–297, March 2007. ISSN 1076-9757. doi: 10.1613/jair.2096. URL https://www.jair.org/index.php/jair/article/view/10489.

Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Classification with Costly Features Using Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 3959–3966, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33013959. URL https://ojs.aaai.org/index.php/AAAI/article/view/4287. Number: 01.

Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Classification with costly features in hierarchical deep sets. *Machine Learning*, May 2024. ISSN 1573-0565. doi: 10.1007/s10994-024-06565-4. URL https://doi.org/10.1007/s10994-024-06565-4.

Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. Rethinking Formal Models of Partially Observable Multiagent Decision Making, September 2021. URL http://arxiv.org/abs/1906.11110. arXiv:1906.11110 [cs].

Lucas Lehnert, Sainbayar Sukhbaatar, DiJia Su, Qinqing Zheng, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond A*: Better Planning with Transformers via Search Dynamics Bootstrapping, April 2024. URL http://arxiv.org/abs/2402.14083. arXiv:2402.14083 [cs].

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. URL http://arxiv.org/abs/1312.5602. arXiv:1312.5602 [cs].

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, May 2017. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aam6960. URL https://www.science.org/doi/10.1126/science.aam6960.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch, July 2017. URL http://arxiv.org/abs/1707.06170. arXiv:1707.06170 [cs, stat].

Simon Ramstedt and Christopher Pal. Real-Time Reinforcement Learning, December 2019. URL http://arxiv.org/abs/1911.04448. arXiv:1911.04448 [cs, stat].

Aviv Rosenberg, Assaf Hallak, Shie Mannor, Gal Chechik, and Gal Dalal. Planning and Learning with Adaptive Lookahead, January 2023. URL http://arxiv.org/abs/2201.12403. arXiv:2201.12403 [cs].

Stuart Russell and Eric H. Wefald. *Do the Right Thing: Studies in Limited Rationality*. The MIT Press, January 2003. ISBN 978-0-262-28277-2. doi: 10.7551/mitpress/2474.001.0001. URL https://direct.mit.edu/books/book/2747/Do-the-Right-ThingStudies-in-Limited-Rationality.

Martin Schmid, Matej Moravčík, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, G. Zacharias Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Student of Games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances*, 9(46):eadg3256, November 2023. doi: 10.1126/sciadv.adg3256. URL https://www.science.org/doi/10.1126/sciadv.adg3256. Publisher: American Association for the Advancement of Science.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL https://www.nature.com/articles/s41586-020-03051-4. Number: 7839 Publisher: Nature Publishing Group.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

L. S. Shapley. Stochastic Games*. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, October 1953. doi: 10.1073/pnas.39.10.1095. URL https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095. Publisher: Proceedings of the National Academy of Sciences.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961%7D. Publisher: Nature Publishing Group.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, December 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/10.1126/science.aar6404. Publisher: American Association for the Advancement of Science.

Samuel Sokota, Hengyuan Hu, David J. Wu, J. Zico Kolter, Jakob Nicolaus Foerster, and Noam Brown. A Fine-Tuning Approach to Belief State Modeling. October 2021. URL https://openreview.net/forum?id=ckZY7DGa7FQ.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. MIT Press, November 2018. ISBN 978-0-262-35270-3. Google-Books-ID: uWV0DwAAQBAJ.

David Sychrovský, Michal Šustr, Elnaz Davoodi, Michael Bowling, Marc Lanctot, and Martin Schmid. Learning not to Regret, February 2024. URL http://arxiv.org/abs/2303.01074. arXiv:2303.01074 [cs].

Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, volume 2011, pp. 674–679, 2011. URL https://www.academia.edu/download/70000255/Bounded_Suboptimal_Search_A_Direct_Appro20210920-6507-1acff16.pdf.

Pierre Thodoroff, Wenyu Li, and Neil D. Lawrence. Benchmarking Real-Time Reinforcement Learning. In *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*, pp. 26–41. PMLR, October 2022. URL https://proceedings.mlr.press/v181/thodoroff22a.html. ISSN: 2640-3498.

Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski. Reactive Reinforcement Learning in Asynchronous Environments. *Frontiers in Robotics and AI*, 5, June 2018. ISSN 2296-9144. doi: 10.3389/frobt.2018.00079. URL https://www.frontiersin.org/articles/10.3389/frobt.2018.00079. Publisher: Frontiers.

A Related Works

Of course, it is not a novel insight to say that MDPs ignore decision-making time. This mismatch in RL has been noted before(Ramstedt & Pal, 2019; Travnik et al., 2018; Thodoroff et al., 2022). In addition, any researcher of decision-time planning algorithms knows that time matters, at least implicitly – no one creates a search algorithm that takes a trillion years to run and then sits back, satisfied. However, the aspect in which search algorithms use more or less time to search is typically (A) controlled by hand-crafted heuristics, not learned parameters or (B) simplistic and not contextual (either a fixed amount of search per decision, or a predetermined amount of search for each decision).

The book *Do the Right Thing: Studies in Limited Rationality* (Russell & Wefald, 2003) states a view of intelligence and decision-time planning algorithms similar to the one stated here. However, the idea that time/resource cost should be considered a fundamental aspect of agents is still underappreciated by researchers of reinforcement learning and decision-time planning algorithms, so we repeat it here, in a post deep-learning world.

Anytime search is a term for a search algorithm that can be stopped at any time, whereupon a solution will be output. If the algorithm is allowed to run for longer, the quality of the returned solution can get better, but not worse. (Hansen & Zhou, 2007). Most methods are not designed to explicitly optimize for solving time. Bounded Suboptimal Search (Thayer & Ruml, 2011) does do this, but attempts to find any solution within a given suboptimality bound, instead of being able to optimize for a more general class of time-optimality tradeoffs.

In this paper, we base our experiments and discussions on the existing search algorithm Thinker (Chung et al., 2023). Other algorithms are similar to Thinker and could potentially be modified in the same way: e.g. MCTSNets (Guez et al., 2018) and Imagination-based Planner Pascanu et al. (2017).

Many previous works have explored using less compute by training neural nets: (Banino et al., 2021; Janisch et al., 2024; 2019; Rosenberg et al., 2023; Lehnert et al., 2024).

B Additional Experiment Details

Experiments were ran using 1 GPU and between 12 and 18 CPU processes each. All experiments used source code from Thinker (Chung et al., 2023). We plan to open-source the modifications to Thinker that enable Dynamic Thinker.

All experiments used the following Thinker flags:
`--actor_learning_rate 0.0003 --entropy_cost 0.0011 --discounting 0.999`

Model-free RL experiments used these additional flags:
`--wrapper_type 1 --has_model false --train_model false --total_steps 120000000`

Other experiments (Thinker Fixed-k, Dynamic Thinker, MCTS) were run using these additional flags:
`--wrapper_type 2 --max_depth 10 --im_cost 0.21 --model_warm_up_n 10001`
`--has_action_seq true --see_x true --model_learning_rate 0.0001`
`--see_real_state true`

MCTS experiments used these additional flags:
`--mcts true --tree_carry false --actor_unroll_len 200 --buffer_traj_len 20`
`--max_depth -1 --auto_res False`

Dynamic Thinker had a cap of 20 on search length..

C Additional Experiment Results

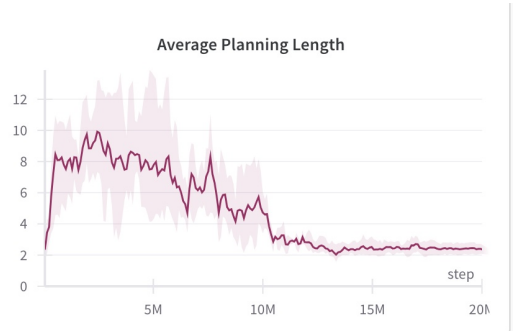
C.1 Planning Steps

Figures 3a and 3b show how the amount of search compute used by Dynamic Thinker changed during the course of training. The “Average" of Figure 3a and the “Max" of Figure 3b are taken over each batch. While the final average planning length is around 2 per decision, most batches contain a planning stage between 10 and 15 steps long. Thus, Dynamic Thinker learned to vary its amount of search depending on context, which means the resulting agent is not in the hypothesis class of the other methods.

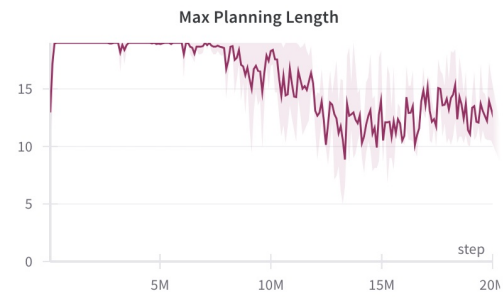
C.2 Extended Results

Each experiment was run for about 3 to 4 hours of wall-time. Model-free RL and MCTS have a higher rate of training steps per wall-time, so they ran for much more than the 2e7 steps shown in Figures 1 and 2. Here, in Figures 4 and 5, we present untruncated versions of the plots.

In Figure 5 it is more evident that the other tree search runs with high values of k achieve higher raw (*time-independent*) returns than Dynamic Thinker, but Dynamic Thinker still achieves the highest



(a) Average Number of Planning Steps in a Training Batch



(b) Maximum Number of Planning Steps in a Training Batch

Figure 3: Dynamic Thinker amount of planning during training

timed returns. This demonstrates that Dynamic Thinker's superior performance is due to learning a superior trade-off between time-independent returns and planning costs, not simply by achieving higher time-independent returns.

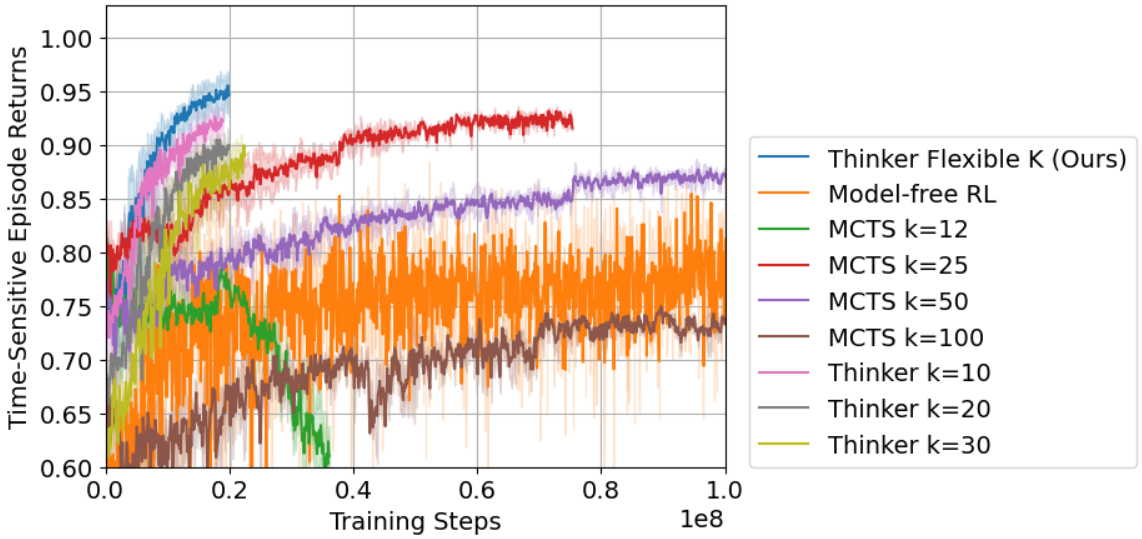


Figure 4: Timed returns during training

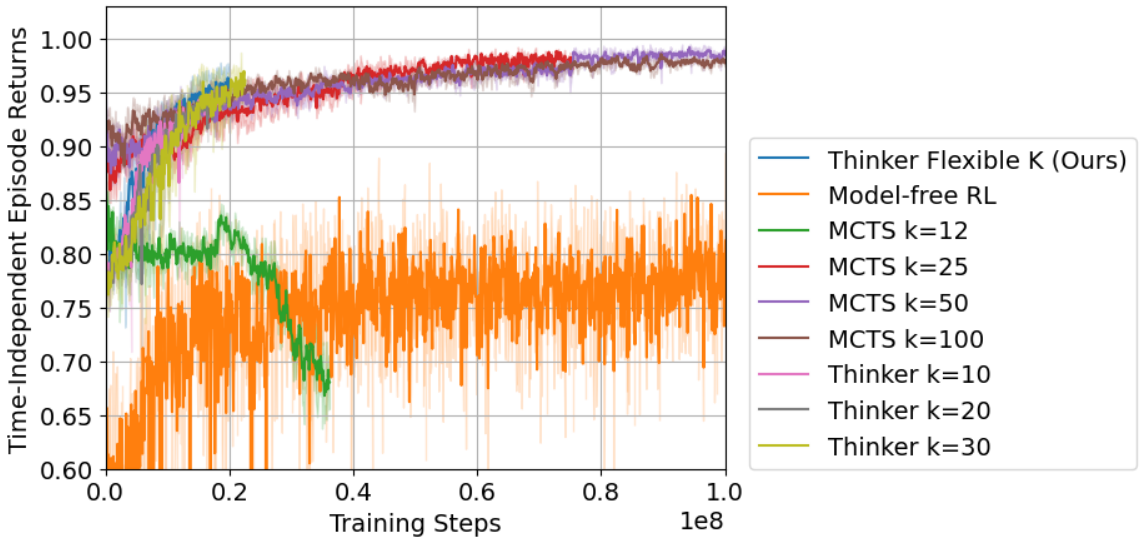


Figure 5: Time-independent returns during training