Variable-Length Diverse Planning Monte Carlo Tree Search

Nathan Amoussou, Denis Pallez, Florent Jaillet, and Jean-Paul Comet

Université Côte d'Azur, CNRS, I3S, France
denis.pallez@univ-cotedazur.fr

Abstract. In the context of sequential decision-making problems, Monte Carlo tree search (MCTS) is a state-of-the-art planning algorithm. It is responsible for the improvement of many computer games (discrete domain) but also for real-world problems involving continuous action spaces. The focus of MCTS is on the analysis of the most promising solutions, expanding the search tree based on random sampling of the search space. When considering diverse planning using continuous-action tree search, existing methods have been validated in a very specific domain assuming fixed-length plans where the Euclidean distance was used as a diversity measure. In this paper, we generalize previous methods for obtaining variable-length plans by introducing the use of an archive that extracts candidate plans online based on a Dynamic Time Warping (DTW) diversity score. We compare the new methods using four reinforcement learning (RL) benchmarks from the Gymnasium and PettingZoo standard environments. Our conclusions are solid and largely consistent with recent results obtained in an all too specific field.

Keywords: Discrete and Continuous Monte Carlo Tree Search · Diverse Planning · Dynamic Time Warping · Gymnasium · PettingZoo

1 Introduction

MCTS is a general decision-time planning algorithm initially designed to improve the computer game of Go [8]. The core idea of MCTS is to build a search tree incrementally, with nodes representing states of the environment and edges representing actions taken from one state to a successor state [3]. However, vanilla MCTS is primarily used to identify a single high-quality plan. In contrast, many real-world applications benefit from a range of diverse alternatives to ensure robustness against uncertainty. Consider, for example, a robot navigating an obstacle-filled environment. While the direct route may seem optimal, having access to alternative paths—such as detours along the left or right edges—ensures resilience when the preferred route is blocked or impractical. This concept is formalized as *diverse planning* [1]. Recent work [14] has proposed and compared four specialized MCTS variants to produce diverse plans but these methods were validated on a domain that is too specific (bioinformatics) and only considering plans with a fixed, low number of actions (i.e. a *fixed horizon*). In this context,

the plans were time-aligned, enabling a diversity measure to be calculated based on the Euclidean distance. This assumption does not apply to standard RL benchmarks, in which the number of actions can vary between different plans. In that case, Euclidean distance is then ill-defined.

In this paper, we address the previous gap through three key contributions: (1) we move from a fixed horizon to a *variable-length* horizon, (2) we provide a principled generalization of MCTS strategies to standard RL benchmarks, resolving key algorithmic ambiguities regarding plan extraction, and (3) we conduct a systematic empirical comparison, showing robust trends across different environments.

The remainder of this paper is organized as follows. First, we review the background of Monte Carlo tree search in discrete and continuous domains in section 2. We then explore recent works on diverse planning mechanisms and address their limitations in section 3, introducing a variable-length diverse planning framework based on DTW score. Section 4 describes benchmark environments, the tuning procedure of the tested algorithms and the experimental protocol. The results are reported and analyzed in section 5. Finally, section 6 discusses the main takeaways, limitations, and directions for future work.

2 Background

2.1 Vanilla Monte Carlo Tree Search

We frame the planning problem as a deterministic Markov decision process [16], defined by the tuple (S, A, T, R) where S is the set of states, A is the set of actions, $T : S \times A \to S$ is the deterministic transition function that maps a state-action pair to a successor state, and $R : S \times A \to \mathbb{R}$ is the reward function. A plan, denoted $\pi = (a_0, a_1, \ldots, a_{n-1})$, is a finite sequence of actions. Executing a plan from an initial state s_0 generates a finite sequence of states $(s_1, \ldots, s_n)$. The main objective is to find a plan of n actions that maximizes a reward, which could be expressed cumulatively as the sum of the rewards received along the sequence, $\sum_{t=0}^{n-1} R(s_t, a_t)$, or simply the reward of the final state, $R(s_{n-1}, a_{n-1})$, for example.

To find such a plan, MCTS builds a search tree by iterating through four steps. (1) Starting from the initial state s_0 (i.e. the root node of the tree), a *tree policy* is used to identify the most interesting state in the tree (*selection*), balancing between exploration and exploitation. This state must have at least one untried action. (2) From this state, one of these untried actions is randomly taken, resulting in a new state added as a node to the tree (*expansion*). (3) A *default policy* is applied from the expanded state to a terminal state in order to estimate its reward (*simulation* or *rollout*). Finally, (4) as the tree keeps statistics for each state (i.e. *visit count* as the number of times a node leads to a simulation), the estimated reward is propagated to all ancestor nodes between the current state and the root (*backpropagation*) and all steps are repeated until a certain budget runs out [3].

MCTS is particularly relevant when an explicit model of the dynamics and rewards is unavailable (i.e. the T and R functions) and only a black-box simulator can be queried. The performance of MCTS relies heavily on the tree policy [3]. The most common is the Upper Confidence bound for Trees (UCT) [11], which has been used to reach human master level in the game of Go (MoGo [9]). UCT selects the action a from state s that maximizes:

$$\mathrm{UCT}(s,a) = Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{1}$$

where $Q(s,a)$ is the estimated reward value of the action, $N(s)$ and $N(s,a)$ are respectively the visit counts of the parent node and of the specific action, and C is a constant controlling the degree of exploration which is highly problem-dependent. Several classical enhancements improve the quality of value estimates by sharing rollout information across related parts of the tree. The All-Moves-As-First (AMAF) heuristic [2] updates the statistics of actions that appear later in a simulation as if they had been played earlier, thereby providing faster (but potentially biased) estimates. Rapid Action Value Estimation (RAVE) [10] linearly combines UCT with AMAF to obtain rapid yet biased action-value estimates. Generalized RAVE (GRAVE) [4] further reduces this bias by borrowing AMAF statistics from a suitably chosen ancestor node, which tends to provide more reliable estimates near the leaves.

2.2 Continuous Monte Carlo Tree Search

For each state, eq. (1) requires every possible action to be simulated at least once before deeper levels of the tree are explored. However, this does not hold for a large-scale or infinite number of candidate actions for each state (e.g. a continuous domain). In that case, Progressive Widening (PW) can be used [6]. For each state s, the set of actions already explored at that state, $C(s)$, is stored. From this state, a new action and its consequent state are explored only if the current number of already explored actions $|C(s)|$ remains below a capacity that grows with the visit count $N(s)$, formalized as:

$$N(s)^w \geq |C(s)| \tag{2}$$

where $w \in (0,1)$ is a problem-dependent parameter that restrains the number of actions allowed in s. If w is close to 0, few actions are considered for s, resulting in a narrower, deeper tree that reuses a few actions. Conversely, if w is close to 1, $N(s)^w$ grows almost linearly with $N(s)$, resulting in the exploration of many distinct actions and a wider tree that emphasizes exploration. However, w must remain less than 1; otherwise, the number of authorized actions will increase faster than the number of times s is visited. This will lead to new actions being explored almost every time, rather than the same ones being tested several times to determine their effectiveness.

When using AMAF-based heuristics in continuous action spaces, the probability of re-sampling the same action is essentially zero, meaning raw AMAF

counts become uninformative. Continuous RAVE (cRAVE) [7] addresses this issue by replacing exact matches with kernel-based smoothing, sharing AMAF statistics across similar state-action pairs via Gaussian convolutions. GRAVE was also adapted to continuous domain (cGRAVE) by [15]. However, this technique has been validated only through bioinformatics benchmarks for identifying gene regulatory network parameters. Our work aims to validate it on more standard benchmarks.

3 Methods

While MCTS primarily focuses on converging toward a single optimal solution, many applications require a portfolio of alternatives to ensure robustness against uncertainty. In this section, we review and build upon different strategies for computing diverse plans.

3.1 Diverse Multi-Armed Bandit MCTS (DMAB)

To bias the MCTS search towards finding diverse plans, [1] formalized the *diverse top-k planning* problem, where the goal is to identify a set of k plans that are optimal in terms of both quality and diversity. The authors defined the *quality* Q_{plan} of a plan π as the normalization of its quality based on the regret of its constituent actions, formalized as $Q_{\text{plan}}(\pi) = \prod_{i=0}^{n-1} \frac{Q(s_{i+1})}{\max_{s' \in C(s_i)} Q(s')}$, where $Q(s)$ is the estimated reward value of the tree node containing the state s. This value is typically less than 1, as diverse plans often include suboptimal actions. Moreover, they define the distance (referred to as *diversity*) $D(\pi, \mathcal{P})$ of a plan π with respect to a set of plans contained in an archive $\mathcal{P}$ as the minimum pairwise distance between π and any plan in $\mathcal{P}$: $D(\pi, \mathcal{P}) = \min_{\psi \in \mathcal{P}} \delta(\pi, \psi)$, with δ a user-defined distance metric (e.g. the Euclidean or Jaccard distances). When the search budget is exhausted, the diverse top-k plans are extracted using a best-first traversal of the tree ordered by plan quality. Each plan that satisfies the minimum quality (q) and diversity (d) thresholds is added to $\mathcal{P}$. Once $\mathcal{P}$ reaches size k, if a newly found plan π has the same quality as the lowest-quality plan $\psi \in \mathcal{P}$ but contributes to a greater diversity, π replaces ψ to maximize the set's diversity without reducing its quality [1].

However, relying solely on post-hoc filtering is often inefficient as the tree naturally converges toward a single optimum. In order to increase diversity, [1] modified the tree policy used during the search by replacing eq. (1) with:

$$\text{DMAB}(s, a) = \text{UCT}(s, a) + D(\pi_a, \mathcal{P}) \tag{3}$$

where π_a represents the partial plan whose corresponding state sequence extends from s_0 to the successor state $T(s, a)$.

Improvements. The issue with this formula is the lack of weights associated with the UCT and diversity values as is the case between exploration and exploitation in eq. (1). Therefore, we propose the new formula DMAB2 which introduces

a new parameter C_{div} balancing DMAB between standard UCT and explicit diversity, expressed as:

$$\text{DMAB2}(s, a) = \text{UCT}(s, a) + C_{\text{div}} \times D(\pi_a, \mathcal{P}) \tag{4}$$

But identifying the value of this weight is problem-dependent and non-trivial. Another possibility is to normalize UCT and diversity values.

3.2 Diverse Planning Multi-Objective MCTS (DPMO)

One way to avoid defining weights between two conflicting objectives, such as quality and diversity, can be achieved by reformulating the weighted sum optimization (eq. (3)) into a multi-objective optimization in which both scores are optimized independently. This is exactly what [14] proposed with DPMO based on a multi-objective variant of MCTS [19]. However, the alternative proposed by [5] can also be employed, as it is more efficient. In that case, the reward value associated with each node of the tree is a vector $r_a = \big(Q_{\text{plan}}(\pi),\, D(\pi, \mathcal{P})\big)$. In order to compare several vectors, the relation of dominance ($\prec$) is used for keeping *non-dominated* rewards of the current tree, saved in a new archive $\mathcal{X}$, that represents all possible trade-offs between quality and diversity. A vector dominates another if it is no worse in all scores and strictly better in at least one. It is important to note that the non-dominated rewards contained in $\mathcal{X}$ are not necessarily the rewards of the plans contained in $\mathcal{P}$ because this archive contains complete plans while $\mathcal{X}$ contains rewards of explored partial plans. However, the strategy outlined by [14] is not clear in its implementation, as the authors do not explain when and how to extract plans of $\mathcal{P}$. Since rewards are now represented as vectors, the tree policy (eq. (1)), originally defined for scalar rewards, must be adapted. Instead of selecting from a state s the most promising action a based on a linear sum of exploration and exploitation scores, DPMO selects the action a that maximizes the most the *hypervolume* indicator of $\mathcal{X}$ if chosen, out of all possible actions from s. The hypervolume is a scalar metric used to compare sets of non-dominated rewards by measuring the size of the objective space they cover.

Improvements. [1] provided a post-hoc approach for extracting diverse plans, but this is insufficient for MCTS diversity variants that require access to an archive during the search. [14] was obliged to use an online adaptation, but did not describe it in their work. We propose in algorithm 1 a formal adaptation of [1] approach that allows the extraction of plans during the search. We maintain an archive $\mathcal{P}$ of capacity k that only accepts candidate plans that satisfy both a quality threshold $q \in [0, 1]$ and a diversity threshold $d \in [0, 1]$ (line 2). Instead of extracting diverse plans post-hoc, plans are evaluated incrementally after the backpropagation phase (section 2.1). The algorithm is only used when the expanded node corresponds to either a terminal node or a node that reached the fixed horizon. Once the archive is full, a new candidate will replace an existing plan only if it improves both the quality q (line 7) and the diversity of the worst plans of $\mathcal{P}$ (lines 9–18). To do so, the diversity is first initialized to its best value

Algorithm 1: Add plan online$(\pi, Q_{\text{plan}}(\pi), \mathcal{P}, q, d, D)$

Input : Add candidate plan π of quality $Q_{\text{plan}}(\pi)$ to archive $\mathcal{P}$ of size k, using quality q and diversity d thresholds, diversity metric $D(\cdot, \mathcal{P})$
Output: Updated archive $\mathcal{P}$

1 $D_\pi \leftarrow D(\pi, \mathcal{P})$
2 **if** $Q_{plan}(\pi) < q$ ***or*** $D_\pi < d$ **then** ▷ `if` Q `or` D_π `below thresholds`
3 **return** $\mathcal{P}$ ▷ `do not add` π `to` $\mathcal{P}$
4 **else if** $|\mathcal{P}| < k$ **then** ▷ `if` $\mathcal{P}$ `is not full`
5 **return** $\mathcal{P} \cup \{\pi\}$ ▷ `add` π `to` $\mathcal{P}$
6 $q_{\min} \leftarrow \min\{\, Q_{\text{plan}}(\psi) \mid \psi \in \mathcal{P} \,\}$ ▷ `worst quality in` $\mathcal{P}$
7 **if** $Q_{plan}(\pi) < q_{\min}$ **then**
8 **return** $\mathcal{P}$
9 $\mathcal{W} \leftarrow \{\, \psi \in \mathcal{P} \mid Q_{\text{plan}}(\psi) = q_{\min} \,\}$ ▷ `extract worst plans of` $\mathcal{P}$
10 $\psi_{\text{worst}} \leftarrow \text{random}(\mathcal{W})$; $D_{\text{worst}} \leftarrow 1.0$ ▷ `choose one randomly`
11 **foreach** $\psi \in \mathcal{W}$ **do** ▷ `worst plan` ψ_{worst} `in quality & diversity`
12 $\mathcal{P}_{\setminus\psi} \leftarrow \mathcal{P} \setminus \{\psi\}\,; D_\psi \leftarrow D(\psi, \mathcal{P}_{\setminus\psi})$
13 **if** $D_\psi < D_{worst}$ **then**
14 $D_{\text{worst}} \leftarrow D_\psi\,; \psi_{\text{worst}} \leftarrow \psi$
15 $\mathcal{P}' \leftarrow \mathcal{P} \setminus \{\psi_{\text{worst}}\}$ ▷ `new archive without worst`
16 $D'_\pi \leftarrow D(\pi, \mathcal{P}')$ ▷ `new diversity with` π
17 **if** $D'_\pi > D_{worst}$ **then**
18 $\mathcal{P} \leftarrow \mathcal{P}' \cup \{\pi\}$ ▷ `replace` ψ_{worst} `with` π `in` $\mathcal{P}$
19 **return** $\mathcal{P}$

(line 10). This algorithm will be used with all other MCTS variants improved in this article.

3.3 Lock-MCTS (Lock)

In addition to multi-objective optimization, [14] proposed a specific inhibition strategy that biases the tree policy by ignoring branches of the tree corresponding to plans already added to the archive. This strategy, termed *Lock*, favours searching for unexplored branches by pruning plans from the tree that have reached the horizon. This kind of strategy seems possible particularly when the planning horizon is short and there is sufficient budget to try out alternative plans, as was the case in their area of application. If not, the algorithm may fail to build at least one good plan, and behave as the vanilla MCTS variant.

Moreover, an ambiguity appears on what is really locked in the proposed strategy. We need to decide whether to block the first action of each plan in the archive, or all the actions it contains. This question is detailed in the next paragraph.

Improvements. In our implementation of the Lock strategy, we decided to inhibit only the initial action of each archived plan (i.e., the action leaving the root).

Locking every action in the sequence could prevent the discovery of new plans, particularly if a specific action acts as an *isthmus*—a unique bridge required to reach the horizon.

3.4 Diverse Progressive Widening MCTS (DivPW)

Another idea from [14] is to adapt the PW mechanism (eq. (2)) to a context of diversity called *DivPW*. Rather than limiting the number of actions from a state s based solely on its visit count $N(s)$, DivPW also limits the number of possible actions by considering only those that are not present in the archive at the corresponding depth, denoted $C_{\text{DivPW}}(s)$. Equation (2) is replaced by $N(s)^w \geq |C_{\text{DivPW}}(s)|$.

To verify whether an action from s is present in the archive, the Euclidean distance between this action and each action stored at the same depth is calculated. If this distance is less than a threshold ε, the actions are considered identical. This ε-check is essential in continuous domains, for both scalar and vector actions, because exact equality is rare; without it, two actions with an infinitesimal difference would be treated as distinct, and we would rarely ever find two exactly matching actions.

[14] used the same Euclidean distance to also compute distances between plans. However, this metric can only be used for plans of the same length as it only compares objects of the same length. It becomes ill-suited when they differ in length, which is often the case in general RL environments. We explore an alternative distance metric in the next paragraph.

Improvements. To measure the distance between plans of different lengths, we propose the use of Dynamic Time Warping (DTW) [20], an algorithm from time series analysis originally designed to align temporal sequences of varying speeds for speech recognition.

DTW aligns two sequences of actions $\pi = (a_1, \ldots, a_n)$ and $\pi' = (a'_1, \ldots, a'_m)$ by finding a *warping path* $\omega = ((i_1, j_1), \ldots, (i_L, j_L))$ with $(i_1, j_1) = (1, 1)$ and $(i_L, j_L) = (n, m)$, where each pair (i_k, j_k) matches action a_{i_k} with a'_{j_k}. This path is monotone ($i_k \leq i_{k+1}$ and $j_k \leq j_{k+1}$) and continuous ($(i_{k+1} - i_k,\, j_{k+1} - j_k) \in \{(1,0), (0,1), (1,1)\}$), and allows an action from π to match one or several actions from π', and vice versa. DTW selects the warping path ω that minimizes $\sum_{k=1}^{L} \rho(a_{i_k}, a'_{j_k})$, where ρ is a local distance metric between actions (e.g., Euclidean distance for continuous actions or $\rho(a, a') = 0$ if $a = a'$, 1 otherwise in a discrete grid world), using the following dynamic programming formula: $\text{DP}(i, j) = \rho(a_i, a'_j) + \min\{\text{DP}(i-1, j), \text{DP}(i, j-1), \text{DP}(i-1, j-1)\}$, with $\text{DP}(0, 0) = 0$ and $\text{DP}(i, 0) = \text{DP}(0, j) = +\infty$ for $i, j > 0$. The DTW distance between π and π' is then defined as $\text{DTW}(\pi, \pi') = \text{DP}(n, m)$. As this raw distance value lies in the interval $[0, \infty)$, it cannot be used directly with the online plan extraction algorithm (alg. 1) described above which assumes diversity values in $[0, 1]$. Therefore, it could be normalized as $\delta_{\text{DTW}}(\pi, \pi') = \frac{\text{DTW}(\pi, \pi')}{\text{DTW}(\pi, \pi') + \lambda}$, where $\lambda > 0$ is a scaling constant that controls how strongly large raw DTW values are compressed: $\delta_{\text{DTW}} = 0.5$ when $\text{DTW} = \lambda$. The DTW diversity between π and

an archive of plans $\mathcal{P}$ is then derived as the minimum pairwise DTW distance of π to any plan in $\mathcal{P}$:

$$D_{\text{DTW}}(\pi, \mathcal{P}) = \min_{\psi \in \mathcal{P}} \delta_{\text{DTW}}(\pi, \psi) \tag{5}$$

This diversity metric will be used with every other MCTS variant described previously.

4 Experimental Setup

In this section, we describe the experimental validation setup used to assess the viability and performance of the improved MCTS variants detailed above. We introduce our protocol in section 4.1, then present the benchmark environments used in section 4.2, and finally describe our automatic hyperparameter tuning procedure in section 4.3.

4.1 Evaluation Protocol

The diversity variants presented above are extensions of the underlying MCTS tree policy designed to improve the diversity of generated plans. To align with the original experimental validation in [14], we implemented all these variants using cGRAVE as the common base search heuristic. However, to move towards a more generalized setting (including plans of varying length), we selected four new RL benchmarks representing classical RL continuous-control tasks. We detail these environments in section 4.2. For each environment and each algorithm, we performed an MCTS search phase with a budget ranging from 25,000 to 50,000 iterations, depending on the complexity of the environment. During each phase, every 100 iterations, we recorded the *average archive diversity* (AAD) defined as the mean pairwise DTW distance between all plans in the archive $\mathcal{P}$, and the *average archive size* (AAS) which counts the number of plans stored in $\mathcal{P}$. This archive was filled online using algorithm 1 with the DTW diversity metric (eq. (5)). Regarding the plan quality metric, we did not rely on the equation defined in section 3.1 because it can be unstable in environments with negative rewards, which is a common occurrence in general RL benchmarks (e.g. environments that reward the proximity to a goal with a score in $(-\infty, 0)$, see [12]). We did not use the problem-specific quality metric from [14] either, as by definition, it contradicts our pursuit of generalization. Therefore, as an alternative, we utilized the simple *reward sum quality*. This metric sums the environment rewards r obtained at each step t of the plan π: $Q_{\text{sum}}(\pi) = \sum_{t=0}^{n-1} r_t$. To ensure statistical significance, we repeated each phase 30 times using a different environment seed. This results in a total of 480 experimental runs (4 variants $\times$ 4 environments $\times$ 30 seeds).

4.2 Environments

For the evaluation protocol, we used four continuous control environments from Gymnasium [18] and PettingZoo [17] as benchmarks depicted in fig. 1. First,

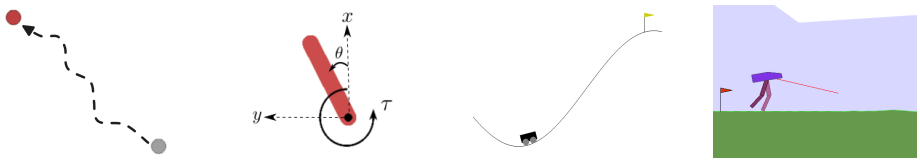
we selected the basic Multi-Agent Particle Environment (MPE) Simple Continuous [12] (hereafter referred to as *Simple*, fig. 1a). This is a minimal 2D navigation task in which a single agent must reach a landmark within a default horizon of 25 steps. This is achieved using a 5-dimensional (continuous) action vector, which corresponds to the four cardinal moves and a no-operation move. It provides a dense reward based on Euclidean proximity to the goal, allowing an MCTS search budget of 25,000 to suffice for all algorithms.

As a secondary fairly standard environment, we also considered Gymnasium Pendulum-v1 (hereafter *Pendulum*, fig. 1b). Here, the agent must swing up and stabilize an inverted pendulum by applying a 1-dimensional torque action over a horizon of 200 steps. The reward is dense and primarily penalizes deviation from the upright configuration, angular velocity, and control effort. This is the only environment for which we increased the MCTS iteration budget to 50,000 because the algorithms did not seem to converge after 25,000 iterations.

To present a greater challenge to the MCTS variants, we moved on to the more complex MountainCarContinuous-v0 environment (hereafter *MountainCar*, fig. 1c). It introduces a harder exploration profile: an underpowered car must build up enough momentum to reach the goal at the top of a hill. Actions are 1-dimensional forces in $[-1, 1]$, episodes reach horizon after 999 steps, and the base reward combines a per-step control penalty with a large bonus when the goal position is reached. As reaching the horizon was impossible within a tractable MCTS budget, we manually reduced it to 50, enabling algorithm 1 to trigger and extract plans. With this modification, a budget of 25,000 iterations proved sufficient.

Finally, the even more complex BipedalWalker-v3 (hereafter *BipedalWalker*, fig. 1d) stresses higher-dimensional control: a 4-joint walker is driven by a 4-dimensional continuous action vector to traverse uneven terrain. The reward encourages forward progress while penalizing actuation effort, with a large negative penalty if the robot falls. The standard horizon is at 1600 time steps. As with MountainCar, we also deliberately capped the horizon at 50 steps, and used an MCTS budget of 25,000 iterations.

Even when the horizon was modified in complex environments, all MCTS variants failed to reliably guide the agent towards the goal. They only allowed algorithm 1 to extract plans that reached the horizon despite being series of seemingly random moves. We hypothesized that this was due to the reward structure of these environments being too sparse. To mitigate this issue, we



(a) Simple (b) Pendulum (c) MountainCar (d) BipedalWalker

Fig. 1: RL benchmarks used from Gymnasium and PettingZoo environments.

Table 1: Environments summary and cGRAVE settings used for the experiments

Environment information				cGRAVE settings			
Env. name	**Category**	**Action dim.**	**Horizon**	w	$bias$	ref	α_{action}
Simple	Standard	5	25	0.25	10^{-3}	16	56
Pendulum	Standard	1	200	0.25	10^{-3}	16	56
MountainCar	Complex	1	50	0.03	10^{-1}	46	40
BipedalWalker	Complex	4	50	0.16	10^{-3}	1	71

implemented custom wrappers that shaped the default environment rewards to the $[0, 1]$ interval, modeled after the structure of Simple and providing gradient signals proportional to the proximity to the goal. This effectively enabled the algorithms to aim towards the goal, albeit still far from reaching it.

We summarize the main characteristics of these environments in table 1.

4.3 Parameter Tuning

MCTS is a hyperparameter sensitive algorithm, i.e. its performance can vary significantly according to the values of its hyperparameters. The cGRAVE backbone we consider exposes four of these: a PW exponent (w, with the same role as in eq. (2)); a bias ($bias$) and a reference count (ref), which manage the influence of ancestor statistics; and an action-kernel bandwidth (α_{action}), which dictates the extent of information sharing between similar continuous actions. This sensitivity to hyperparameters and the non-trivial number of parameters justify the use of an automatic algorithm configuration tool. We used Irace [13], a software that relies on iterated racing procedures to find the most robust hyperparameter settings.

This tool requires the definition of search spaces for each targeted hyperparameter. We used the same as in [14]: $w \in (0.0, 1.0)$; $bias \in \{10^{-15}, 10^{-14}, \ldots, 10^{-1}\}$; $ref \in \{1, \ldots, 100\}$; and $\alpha_{action} \in \{1, \ldots, 100\}$. We also fixed the DivPW action similarity threshold $\varepsilon = 0.1$, and the DMAB diversity constant we introduced with DMAB2 at $C_{\text{div}} = 1.0$ to mimic the behavior of the default DMAB used in the original experiment.

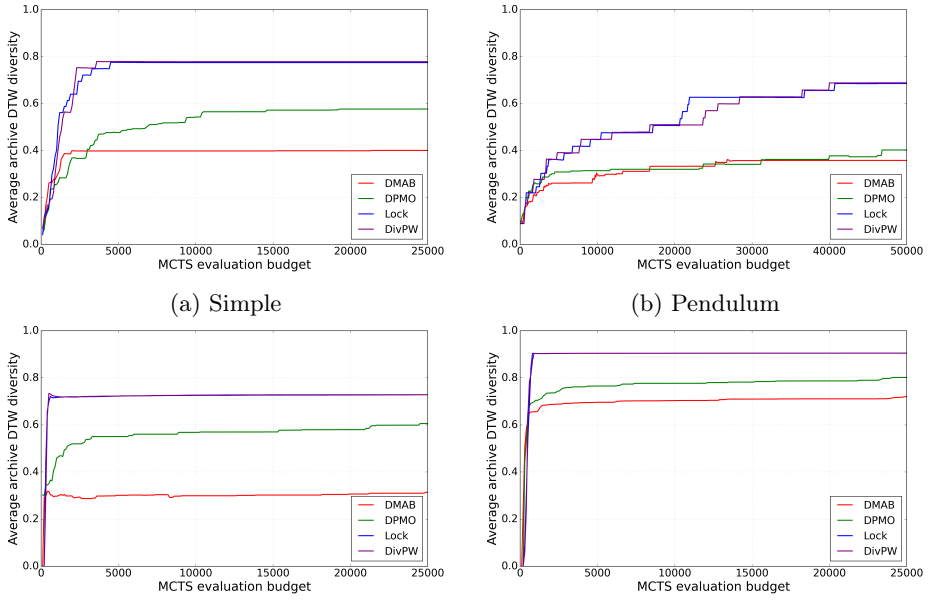
To find values for these hyperparameters, we performed independent Irace runs for each environment, with a budget of 500 algorithm evaluations per run. During these evaluations, the MCTS planner was given a budget of 10,000 iterations. At the end of each MCTS search, a standard UCT exploitation traversal of the tree was performed to identify the optimal plan available. In standard environments (Simple, Pendulum), Irace maximized the reward sum quality of this plan. However, for complex exploration tasks (MountainCar, BipedalWalker), initial results using the same metric were unsatisfactory. The hyperparameters defined highly noisy MCTS behaviors. Therefore, we moved to the *terminal reward quality* as the Irace signal. It evaluates the quality of a plan π solely based on the reward of its final state r_{n-1}: $Q_{\text{last}}(\pi) = r_{n-1}$. Combined with the reward shaping wrappers detailed above, it effectively acts as a distance-to-goal metric,

allowing the algorithms to distinguish between two failed plans by favoring the one that ends physically closer to the objective. The resulting configurations were more robust. Summarized in table 1, they reveal distinct behaviors required by different environments. For instance, MountainCar required a very low w (0.03), indicating that a narrow, deep search is necessary to build the required momentum, whereas Simple favored a broader search (0.25). Note that for Pendulum, we reused the parameters tuned for Simple, as preliminary experiments showed they yielded more stable behavior than its own dedicated tuning run.

5 Results

Building on the experimental setup outlined in the previous section, this section presents the empirical results of the four distinct MCTS variants across the benchmark suite.

First, we analyze how the average archive diversity evolves as the MCTS evaluation budget increases, for each variant in each environment, through fig. 2. This line chart represents AAD on the y-axis as a function of the MCTS evaluation budget on the x-axis. On Simple (fig. 2a), the inhibition-based methods Lock and DivPW rapidly increase diversity and plateau at the highest levels, while DPMO stabilises at an intermediate value and DMAB converges to the lowest diversity. On Pendulum (fig. 2b), absolute diversity is lower overall, but



(a) Simple (b) Pendulum

(c) MountainCar (d) BipedalWalker

Fig. 2: AAD as a function of the MCTS evaluation budget across environments.

the same ranking holds: Lock/DivPW remain consistently above DPMO and DMAB throughout the budget. Here however, DMAB and DPMO proved to be much closer to one another.

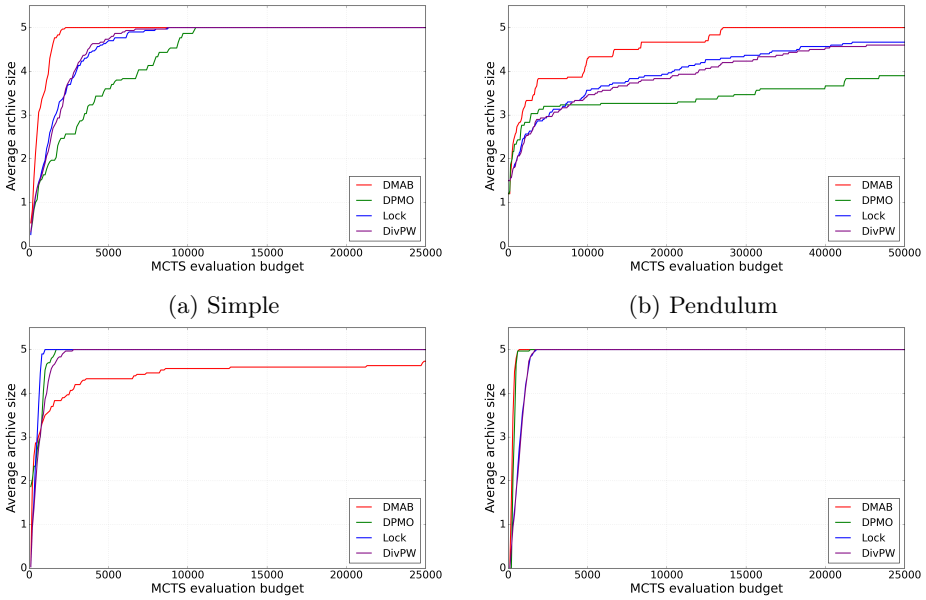
On MountainCar (fig. 2c), Lock and DivPW again dominate and reach high diversity very early, DPMO follows at a moderate level, and DMAB remains substantially behind. Finally, on BipedalWalker (fig. 2d), Lock and DivPW achieve the highest diversity again, with DPMO and DMAB forming a lower, second tier, with DPMO slightly outperforming DMAB. Overall, these curves indicate a consistent pattern across environments: explicit inhibition (Lock, DivPW) tends to be the most effective mechanism for producing highly distinct plan archives under a fixed budget.

To evaluate these observed trends, we conducted a statistical validation campaign. For each environment, we considered the null hypothesis H_0, which states that all algorithms perform similarly in terms of AAD. We first employed the non-parametric Friedman test to challenge this hypothesis. At a significance level of $\alpha = 0.05$, this test rejected H_0 in all four environments, with p-values of 4.74×10^{-17}, 4.55×10^{-10}, 1.15×10^{-13} and 1.47×10^{-15} for Simple, Pendulum, MountainCar and BipedalWalker, respectively. To identify differences between specific algorithm pairs, we then performed a pairwise Wilcoxon signed-rank test with Bonferroni correction. This test did not highlight significant differences between Lock and DivPW in any environment, nor between DMAB and DPMO in the Pendulum environment specifically. In other situations, however, significant differences were observed (table 2).

To complement the diversity trends above, we now examine how quickly each method populates its portfolio by tracking the average archive size over the search budget, via fig. 3. It reports the AAS on the y-axis, as a function of the MCTS evaluation budget on the x-axis. On Simple and Pendulum (figs. 3a and 3b), DMAB is the fastest to populate the archive and is the only variant to reach a full archive on average on Pendulum. The other variants lag behind, DPMO being the clear worst performer. This trend reverses on MountainCar (fig. 3c), where DMAB is the only variant to never reach saturation. In this environment, all other variants exhibit close performances. On BipedalWalker (fig. 3d), no distinct behavior can be attributed to any variant. Overall, these curves reveal a different trade-off to the one observed for diversity. In standard environments, DMAB was the most efficient approach for finding a full (yet

Table 2: Resulting p-values from the pairwise Wilcoxon signed-rank test.

Comparison	**Simple**	**Pendulum**	**MountainCar**	**BipedalWalker**
Lock vs DivPW	**1.00**	**0.35**	**1.00**	**1.00**
DMAB vs DPMO	5.6e-8	**1.00**	2.2e-8	7.3e-5
DMAB vs Lock	1.1e-8	3.6e-4	1.1e-8	1.1e-8
DMAB vs DivPW	1.1e-8	2.8e-4	1.1e-8	1.1e-8
DPMO vs Lock	1.1e-8	2.8e-4	1.4e-5	1.1e-7
DPMO vs DivPW	1.1e-8	2.8e-4	2.8e-6	2.2e-8



(a) Simple (b) Pendulum

(c) MountainCar (d) BipedalWalker

Fig. 3: AAS as a function of the MCTS evaluation budget across environments.

not so diverse) set of solutions, outperforming other variants. In more complex environments, however, its performance was either similar to or worse than that of the other variants. All end-of-budget statistics for diversity and archive size are summarized in table 3.

6 Discussion and Conclusion

The primary objective of this work was to generalize diverse planning strategies within continuous MCTS beyond the specific bioinformatics domain where they were originally proposed. By extending the cGRAVE baseline with online plan extraction and variable-length plan comparison, we successfully deployed and evaluated four distinct variants, DPMO, DMAB, Lock, and DivPW, on standard RL benchmarks.

Overall, our results indicate that explicit diversity constraints (DivPW and Lock) are the most reliable way to obtain diverse plan archives under our generalized setup. Across the benchmarks, DivPW and Lock consistently achieve the highest DTW-based diversity, while DPMO produces more diverse archives than DMAB but remains below the inhibition-based methods in our setting. At the same time, the archive-size curves show that how quickly the archive saturates can vary with the environment: DMAB can populate the archive rapidly in simpler tasks, whereas inhibition-based methods tend to stabilize near a full archive in most environments while prioritizing diversity.

Table 3: AAD and AAS measures summary.

Environment	Planner	AAD ± std	AAS ± std
Simple	DMAB	0.390 ± 0.077	$\mathbf{4.878} \pm 0.741$
	DPMO	0.516 ± 0.161	4.328 ± 1.439
	Lock	0.740 ± 0.164	4.651 ± 1.064
	DivPW	$\mathbf{0.743} \pm 0.167$	4.659 ± 1.058
Pendulum	DMAB	0.322 ± 0.150	$\mathbf{4.564} \pm 1.401$
	DPMO	0.336 ± 0.257	3.401 ± 2.152
	Lock	$\mathbf{0.550} \pm 0.336$	3.991 ± 1.719
	DivPW	0.544 ± 0.341	3.908 ± 1.743
MountainCar	DMAB	0.301 ± 0.133	4.415 ± 1.090
	DPMO	0.560 ± 0.106	4.911 ± 0.489
	Lock	$\mathbf{0.718} \pm 0.073$	$\mathbf{4.924} \pm 0.517$
	DivPW	$\mathbf{0.718} \pm 0.076$	4.875 ± 0.598
BipedalWalker	DMAB	0.697 ± 0.072	$\mathbf{4.957} \pm 0.439$
	DPMO	0.766 ± 0.107	4.939 ± 0.521
	Lock	$\mathbf{0.889} \pm 0.117$	4.871 ± 0.667
	DivPW	$\mathbf{0.889} \pm 0.117$	4.869 ± 0.671

Regarding the multi-objective formulation (DPMO), our experiments do not reveal a dominance of DPMO in this benchmark suite. This is not necessarily contradictory to the original study [14], where performance depended on the problem characteristics rather than showing a strict winner. DPMO does still improve over the linear-combination baseline DMAB, supporting the benefit of separating objectives compared to a single weighted sum.

Finally, while DTW successfully enabled the comparison of variable-length plans, a critical requirement for general RL tasks, it remains a geometric metric operating directly on raw action sequences. This approach assumes that structural difference in actions equates to semantic diversity in outcome, which may not hold in all domains. Future work could investigate alternative dissimilarity measures, such as the Fréchet distance, or more semantic metrics based on learned latent representations of state plans. Coupled with hybridization strategies (e.g., using DivPW to boost the exploration of DPMO), these advances could further bridge the gap between planning and robust decision-making in complex continuous environments.

Acknowledgments

This work has been supported by the French government, through the France 2030 investment plan managed by the Agence Nationale de la Recherche, as part of the "UCA DS4H" project, reference ANR-17-EURE-0004.

References

1. Benke, L., Miller, T., et al.: Diverse, Top-k, and Top-quality Planning Over Simulators. ECAI (2023). https://doi.org/10.3233/FAIA230275

2. Bouzy, B., Helmstetter, B.: Monte-Carlo Go Developments (2004). https://doi.org/10.1007/978-0-387-35706-5_11
3. Browne, C.B., Powley, E., et al.: A Survey of Monte Carlo Tree Search Methods. IEEE Trans. on CIAIG (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
4. Cazenave, T.: Generalized rapid action value estimation. In: IJCAI (2015), https://dl.acm.org/doi/10.5555/2832249.2832354
5. Chen, W., Liu, L.: Pareto Monte Carlo Tree Search for Multi-Objective Informative Planning. In: Robotics: Science and Systems XV (2019). https://doi.org/10.15607/rss.2019.xv.072
6. Couëtoux, A., Hoock, J.B., et al.: Continuous Upper Confidence Trees. LION (2011). https://doi.org/10.1007/978-3-642-25566-3_32
7. Couëtoux, A., Milone, M., et al.: Continuous Rapid Action Value Estimates. In: Asian conference on machine learning (2011), https://proceedings.mlr.press/v20/couetoux11.html
8. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: CG (2006). https://doi.org/10.1007/978-3-540-75538-8_7
9. Gelly, S., Silver, D.: Achieving Master Level Play in 9 x 9 Computer Go. In: AAAI-08 (2008), http://www.aaai.org/Papers/AAAI/2008/AAAI08-257.pdf
10. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence (2011). https://doi.org/10.1016/j.artint.2011.03.007
11. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: ECML (2006). https://doi.org/10.1007/11871842_29
12. Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., Mordatch, I.: Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. Neural Information Processing Systems (NIPS) (2017). https://doi.org/10.48550/arXiv.1706.02275
13. López-Ibáñez, M., Dubois-Lacoste, J., et al.: The irace package: Iterated Racing for Automatic Algorithm Configuration. Operations Research Perspectives (2016). https://doi.org/10.1016/j.orp.2016.09.002
14. Michelucci, R., Comet, J.P., Pallez, D.: Comparing Diverse Planning Strategies with Continuous Monte Carlo Tree Search Applied to Hybrid Gene Regulatory Networks. In: ICTAI (2024). https://doi.org/10.1109/ICTAI62512.2024.00018
15. Michelucci, R., Pallez, D., et al.: Improving continuous Monte Carlo Tree Search for identifying parameters in hybrid gene Regulatory Networks (2024). https://doi.org/10.1007/978-3-031-70085-9_20
16. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming (1994). https://doi.org/10.1002/9780470316887
17. Terry, J.K., Black, B., et al.: PettingZoo: Gym for Multi-Agent Reinforcement Learning (2021). https://doi.org/10.48550/arXiv.2009.14471
18. Towers, M., Kwiatkowski, A., et al.: Gymnasium: A Standard Interface for Reinforcement Learning Environments (2024). https://doi.org/10.48550/arXiv.2407.17032
19. Wang, W., Sebag, M.: Multi-objective Monte-Carlo Tree Search. In: Asian Conference on Machine Learning (2012), https://inria.hal.science/hal-00758379
20. White, G.M., Neely, R.B.: Speech recognition experiments with linear predication, bandpass filtering, and dynamic programming. IEEE Trans. on ASSP (1976). https://doi.org/10.1109/TASSP.1976.1162779