Continuous Monte Carlo Search

Lotfi Kobrosly
LAMSADE
Université Paris Dauphine-PSL
Paris, 75016 France
lotfi.kobrosly@dauphine.eu

Tristan Cazenave
LAMSADE
Université Paris Dauphine-PSL
Paris, 75016 France

Abstract

Planning problems with continuous search and action spaces are common in real-world applications such as robotics or space exploration, whereas most planning algorithms rely on discrete features. Some attempts have been made to handle the continuous aspect especially in Reinforcement Learning, leveraging a Markov Decision Process modeling to handle it. Monte Carlo Tree Search-based methods also rely on this representation but still haven't made full use of some of the variants' potential, which is why we propose Continuous Nested Monte Carlo Search and Continuous Nested Rollout Policy Adaptation and show that they provide promising results on three problems: Map Traveling, Mountain Car Continuous and two instances of Chemical Processes from the `pc-gym` package.

1 Introduction

Operational planning is a central theme in operational research, especially with the need to optimize decisions over time under constraints and, for some settings, uncertainty in events and/or outcomes. Classical approaches, such as linear programming [10], dynamic programming [33], and constraint programming [35], have provided powerful tools for solving such problems. However, as real-world systems have grown in complexity, the need to model sequential decision-making has led to the adoption of probabilistic frameworks. Among these, Markov Decision Processes (MDPs) [26] have emerged as a foundational model, offering a solid base to represent environments where outcomes depend on current states and chosen actions. MDPs formalize the planning problem as the search for a policy that maximizes expected cumulative reward, balancing immediate and long-term gains.

Building on this framework, Reinforcement Learning (RL) [34] has gained prominence as a data-driven approach to solving MDPs when the system dynamics are unknown or too complex to model explicitly. RL methods enable agents to learn policies through interaction with the environment, using trial-and-error and feedback loops. Techniques such as value iteration, policy gradients, Q-learning and various adaptations of Deep Reinforcement Learning [29] have demonstrated strong performance across a variety of domains, from robotics to logistics and energy systems as well as finance [21; 4]. In parallel, Monte Carlo Tree Search (MCTS) has proven rather effective in planning problems with large decision spaces such as games like single-player ones [28] or Go [12; 30], combining sampling-based exploration with tree-structured value estimation to guide decision-making.

While most work in operational research was often centered on discrete state and action spaces, many applications, such as autonomous control [6], robotics [25], and space exploration [19], involve continuous domains. In such settings, both the state space and the action space are continuous, posing significant challenges for traditional methods [5]. Exact solutions become infeasible and even approximate methods must contend with function approximation and efficient exploration.

To handle these particular challenges, we propose two novel algorithms based on variations of MCTS: *Continuous Nested Monte Carlo Search* and *Continuous Nested Rollout Policy Adaptation*. We

19th European Workshop on Reinforcement Learning (EWRL 2026).

test these approaches on different problems and compare them to available baselines to assess and highlight their ability to handle this specific setting.

2 Related Work

The challenge posed by the continuity of the search space in planning or problems is rather significant. To address it, modern approaches integrate function approximators such as neural networks or kernel methods to represent value functions and policies over continuous spaces [15]. Other methods were based on theoretical guarantees and heuristics adaptation for search-based planning [13].

In RL, in addition to space and time discretizations approaches [11] or discrete approximation of the search space as in hippocampal navigation where cells are regions of the search space instead of single points [32], early theoretical attempts employed a fitted Q-iteration method with approximate policy maximization [3]. On the experimental side, continuous fitted value iterations was implemented to handle physical systems [22]. Other works relied on actor-critic methods and deterministic policy gradients to address continuous actions in portfolio management and linear-quadratic control [31; 17], or through Sequential Monte Carlo methods [20]. Clustering actions to be represented as a value function can also help to effectively learn the policy [24].

Similarly, several extensions of MCTS were proposed to handle the continuity of the search space. Since there is no finite number of actions in a given state, the construction of the next states in the search tree needs to be modified. One approach incorporates progressive widening, *i.e*, when a state's number of visits exceeds a certain threshold, a new child state is sampled and added to the tree [37]. For new unseen actions, a segmentation of the action space can be applied and the action-value estimation can occur using a Voronoi Optimistic Optimization [18]. Other extensions adapt variations of MCTS, namely Rapid Action-Value Estimation (RAVE) [12] and its generalized version GRAVE [8], by changing the All Moves As First (AMAF) [1] heuristic computation with a gaussian kernel to neighboring states-actions pairs [23].

While MCTS focuses on state-value estimation, with some addition of actions' relevance in RAVE and GRAVE, other variations tend to leverage the trajectory and/or the actions values such as Nested Monte Carlo Search (NMCS) [7], Nested Rollout Policy Adaptation (NRPA) [27] or its generalized version GNRPA [9]. A Continuous Pareto-NRPA [2] was elaborated to handle multi-objective problems that dwell within a continuous search space, relying on progressive widening to add candidate actions at each state in a similar fashion to the continuous MCTS. Our motivation is then to propose a different variant of this continuous space-adequate version of NRPA and add a continuous NMCS version to handle the problems differently.

3 Continuous Monte Carlo Search

In this work, we present two novel algorithms, each with two variations, which are the adaptations of NMCS and NRPA to continuous state and action spaces.

3.1 Continuous Nested Monte Carlo Tree Search (cNMCS)

Similarly to cMCTS, we derive our algorithm from the Nested Monte Carlo Tree Search algorithm [7]. For a *level 1* cNMCS, In the initial state, We sample uniformly a set with a fixed number of actions, a hyperparameter δ we will refer to as *bandwidth*. Then, we play the next step using one of two possibilities:

- **Rollout cNMCS (cNMCS)**: for each action sampled, we play a random rollout, sampling a valid action uniformly from the action space at each state we encounter. At the end of the rollout, we get the final score and attribute it to the action from the set of actions we devised at the initial state. The action with the best score is then chosen, and we redo this exact same algorithm from the resulting state.
- **Reward cNMCS (cRbNMCS)**: for each action, we compute the corresponding reward associated to each resulting state, and choose the (action, state) couple that provided the best reward and we reiterate from this last state. This variation is only feasible when a reward function is available, either by default or constructed.

For *level 2*, for each action sampled at the initial state and its resulting state, we run a level 1 cNMCS. The score associated with the action is then the one encountered at the end of the run. Then, we choose the action with the best score. This nesting step helps the algorithm explore more of the search space and choose a better suited action at each state. More details are explained in algorithm 1.

Algorithm 1 cNMCS

Input: $s, a, Seq, S_T, \mathcal{A}, L, \delta$: current state, action, states sequence, score, action space, level, bandwidth
Output: Seq, S_T: states sequence, score

$s \leftarrow T(s, a)$ (transition function)
$S_T \leftarrow S_T + R(s, a)$ (reward function)
$Seq \leftarrow Seq + [s]$
if $L = 0$ **then**
  if Rollout cNMCS **then**
    while s not terminal **do**
      $a \sim \mathcal{U}(\mathcal{A})$ (sample a random action)
      $s \leftarrow T(s, a)$
      $Seq \leftarrow Seq + [s]$
      $S_T \leftarrow S_T + R(s)$
    end while
  end if
else
  while s not terminal **do**
    $A \leftarrow list(),\ scores \leftarrow list()$
    for $i \in \{1, .., \delta\}$ **do**
      $a_i \sim \mathcal{U}(\mathcal{A})$ (sample a new action)
      $A \leftarrow A + [a_i]$
      $\emptyset, S_T \leftarrow$ cNMCS$(s, a_i, Seq, S_T, \mathcal{A}, L - 1, \delta)$
      $scores \leftarrow scores + [S_T]$
    end for
    $a \leftarrow getBestAction(A, scores)$
    $s \leftarrow T(s, a)$
    $Seq \leftarrow Seq + [s]$
    $S_T \leftarrow S_T + R(s)$
  end while
end if
return Seq, S_T

3.2 Continuous Generalized Nested Rollout Policy Adaptation (cNRPA)

Similarly to cNMCS, we adapt the Nested Rollout Policy Adaptation [27] algorithm to the continuous aspect of the problems. Depending on each problem, we may need to take the current state into consideration, making the algorithm similar to a Q-learning scheme [36]. For $n_{policies}$, we run a policy-based rollout. The policy would determine the action chosen at each encountered state, instead of the weights of each possible action in discrete NRPA, since there is no longer a finite number. Then, using the last best sequence of actions and states, we adapt the values of the policy as detailed in algorithm 3. We use two approaches to handle the continuity of the action space. Here we detail a level 1 cNRPA:

- **Gaussian cNRPA (GcNRPA)**: since the search space is continuous, the algorithm is bound to see, at almost every step, a new state it has not visited yet. To choose the action there, we use a *gaussian density kernel* on the actions of other visited states, based on the distance between them and the current state as in equation 1. In addition to adapting the saved actions of the visited states, we use a similar kernel to adapt the policy values of states neighboring the ones visited through the last best sequence.
- **By region cNRPA (PbRcNRPA)**: we subdivide the search space into regions. For each region, only **one** action is the most probable. When encountering a state, the policy returns

the action of the region to which the state belongs. The adaptation of the policy value of a region uses the actions chosen for each state encountered that belongs to it. After a set number of visits to a region, it is subdivided for finer policy, as specified in algorithm 2.

$$a \leftarrow \sum_{s' \text{visited}} K(s, s') * \pi(s') \tag{1}$$

To allow further exploration, we sample around the returned action of the policy using a normal distribution. The standard deviation of this distribution decreases with the rollouts to converge to the policy obtained, as shown in equation 2 where π is the policy and s is the current state.

$$a \sim \mathcal{N}(\pi(s), \Sigma^2) \tag{2}$$

Algorithm 2 Subdivide Region

Input: LB, UB: lower bounds list, upper bounds list (must be of same size)
Output: $newRegions$: list of new regions

1: $NB \leftarrow list()$ (new bounds)
2: **for** $i \in \{1, .., size(UB)\}$ **do**
3: $\quad MB \leftarrow \frac{UB[i]+LB[i]}{2}$
4: $\quad NB \leftarrow NB + [\ ((LB[i],\ MB),\ (MB,\ UB[i]))\ ]$
5: **end for**
6: **return** CartesianProduct(NB) (gets all new possible regions)

4 Experimentation environments and results

All algorithms were implemented in `python`. For comparison, we tested our algorithms on three sets of problems and compared them to baselines to assess their performance. For a more statistically significant assessment of the results, we refer the reader to the appendix.

4.1 Map Traveling

To test the algorithms presented above, we devise a map traveling scheme, where we create a number of 2D maps with obstacles, and an agent at a start position tries to reach a destination. Although we can discretize the steps of the agent, we can broaden the map traveling by going beyond this discretization and use continuous position and movements. For this scenario, the movement is comprised of choosing a direction and a step size, *i.e* a two-dimensional vector at each step.

We test the algorithms mentioned above on a number of custom maps of dimensions 100 x 100 with varying degrees of difficulty, *i.e*, number and disposition of obstacles as well as the start point and the destination positions within each map. We compare them to other Monte Carlo-based algorithms, namely MCTS, RAVE and GRAVE and their continuous versions. To ensure that each algorithm terminates, we set the maximum number of steps to 100 but the iteration can stop when the position of the agent x_F is within a radius of 0.5 pixels of the destination d, *i.e*, $||x_F - d|| < 0.5$. By setting the maximum step size to $\lambda = 10$, we compute the score as follows:

$$score = \lambda * n_{steps} + ||x_F - d||$$

When $score < 1000$, the destination was reached in less steps than 100. The best results of 10 runs of each algorithm are shown in tables 2 and 3. The best scores at each level are in bold, and the best overall score for each map is with an asterisk. Algorithms' parameters are shown in table 1

Parameters	δ	$n_{policies}$	Trajectory length	Baselines's $n_{iterations}$
Level 1	25	200	100	10000
Level 2	15	50		

Table 1: Parameters of algorithms used for the Map Traveling problem

Algorithm 3 Adapt policy

Global parameters: K, t_r: gaussian kernel, threshold
Input: $\pi, Seq, Actions, V_N, V_T, A, \alpha$: policy, states sequence, actions sequence, number of visits per region, threshold of visits per region, state space area, learning rate
Output: π

if GcNRPA **then**
  if π not initialized **then**
    for $i \in \{1, .., size(Seq) - 1\}$ **do**
      $\pi(Seq[i]) \leftarrow Actions[i]$
    end for
  else
    for $i \in \{1, .., size(Seq) - 1\}$ **do**
      if $Seq[i] \in \mathcal{D}_\pi$ (visited states) **then**
        $\pi(Seq[i]) \leftarrow \pi(Seq[i]) + \alpha * (Actions[i] - \pi(Seq[i]))$
      else
        $\pi(Seq[i]) \leftarrow Actions[i]$
      end if
    end for
    for $s \in \mathcal{D}_\pi \backslash Seq$ **do**
      $newMove \leftarrow 0$
      for $i \in \{1, .., size(Seq) - 1\}$ **do**
        $newMove \leftarrow newMove + K(s, Seq[i]) * Actions[i] * \delta_{d(s,Seq[i]) \leq t_r}$ (only added if the distance is smaller than the threshold)
      end for
      if $newMove \neq 0$ **then**
        $\pi(Seq[i]) \leftarrow \pi(Seq[i]) + \alpha * (newMove - \pi(Seq[i]))$
      end if
    end for
  end if
else
  $updateVisits(V)$ (update number of visits of region)
  for $region \in V_N$ **do**
    if $V_N(region) > V_T(region)$ **then**
      $NRS \leftarrow subdivideRegion(LB_{region}, UB_{region})$
      for $NR \in NRS$ **do**
        $V_N(NR) \leftarrow V_N(region)$
        $V_T(NR) \leftarrow \frac{A}{getArea(NR)}$
      end for
      Delete $region$ from V_T and V_N
    end if
  end for
  for $region \in V_N$ **do**
    $newMove \leftarrow$ mean$\{Actions[i], \forall i$ where $Seq[i] \in region\}$
    $\pi(region) \leftarrow \pi(region) + \alpha * (newMove - \pi(region))$
  end for
end if
return π

Map n°	1	2	3	4	5	6	7	8
Baselines								
MCTS	1011.0	1011.0	**190.0**	1011.4	1011.0	1011.0	1011.0	1011.0
RAVE	1011.0	1016.7	560.0	1011.4	1011.0	1011.0	1016.4	1011.0
GRAVE	1011.0	1013.0	470.0	1011.4	1011.0	1012.2	1013.6	1011.0
cMCTS	**460.1**	**220.4**	330.2	**380.2**	**220.3**	**610.2**	**440.3**	**280.3**
cRAVE	720.4	470.2	680.3	530.5	570.5	960.5	750.5	590.1
cGRAVE	860.3	500.3	710.2	680.2	540.0	900.2	450.4	480.4
Level 1								
cNMCS	510.2	370.1	570.1	**400.1**	270.2	470.4	480.1	330.3
cRbNMCS	**180.2**	**160.4**	**170.5**	1016.2	**130.3**	**200.5**	140.0	**130.0**
GcNRPA	1012.9	660.5	880.4	1011.8	510.4	1013.0	440.3	900.4
PbRcNRPA	250.5	240.1	260.4	430.3	140.2	210.3	**100.1**	700.4
Level 2								
cNMCS	360.0	190.3	240.2	320.2	230.1	400.1	220.4	200.1
cRbNMCS	160.2	120.1	120.4	170.2	110.2	180.2	110.5	110.2
GcNRPA	**100.2***	**70.2***	**80.0***	**110.3***	**60.2***	**110.2***	**70.4***	**70.0***
PbRcNRPA	130.2	70.2	160.1	220.6	100.4	182.6	123.4	221.8

Table 2: Best scores on maps from 1 to 8 after 10 runs of each algorithm

Map n°	9	10	11	12	13	14	15
Baselines							
MCTS	1011.0	1012.2	1011.0	1011.0	1011.0	1011.0	**1011.0**
RAVE	1011.0	1016.7	1011.0	1011.0	1033.8	1011.0	1015.0
GRAVE	1011.0	1020.1	1011.0	1011.0	1025.5	1011.0	**1011.0**
cMCTS	**450.4**	**470.4**	**410.3**	**250.2**	**740.2**	650.5	1011.6
cRAVE	940.5	1010.7	950.1	460.1	1011.6	740.4	1010.7
cGRAVE	560.4	1013.7	750.4	460.4	880.4	**610.3**	1013.7
Level 1							
cNMCTS	540.4	430.5	470.2	290.3	450.5	**600.3**	**810.4**
cRbNMCTS	140.2	1058.8	1025.2	1025.2	1068.6	1061.5	1032.7
GcNRPA	900.3	1014.3	970.4	470.5	1015.4	1014.7	1018.9
PbRcNRPA	**140.1**	**140.4**	**370.4**	**320.4**	**440.4**	1020.4	1018.3
Level 2							
cNMCS	340.1	310.2	290.1	220.1	400.1	320.1	480.2
cRbNMCS	130.4	160.5	160.1	130.1	200.2	1015.0	1018.0
GcNRPA	**100.3***	**122.9***	**70.4***	**65.1***	**150.1***	**237.8***	**203.3***
PbRcNRPA	110.4	358.7	209.5	119.3	259.8	305.7	644.5

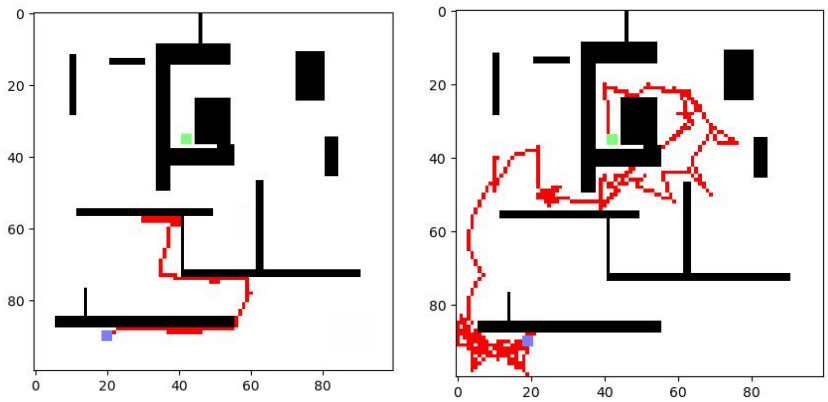
Table 3: Best scores on maps from 9 to 15 after 10 runs of each algorithm

We observe that our algorithms outperform the selected discrete baselines on all problems, and increasing the nesting level provides even better results. This implies that the model these algorithms draw of the search space and actions are more efficient than the baselines. We also see that our algorithms perform similarly to the continuous versions of these baselines, with cMCTS being the hardest to beat, but with one exception. Indeed, at level 1, cRbNMCS is the most proficient for most maps, outperforming even the continuous baselines, despite struggling to reach the destination on harder maps (maps 4 and 10 to 15) as we believe that its reliance on direct rewards from each step restricts its exploration capacity. Figure 1 shows how cRbNMCS gets stuck from the way it operates, while cNMCS allows further exploration. It is also worth noting that most algorithms in level 1 find good results much faster than the discrete baselines, especially cRbNMCS. Higher nesting levels provide better results and we see a better performance than all by **GcNRPA** on all maps, but at the cost of execution time.

4.2 Mountain Car Continuous

OpenAI's Gymnasium[1] offers a wide variety of environments to train and test various Reinforcement Learning and planning algorithms. They range from control mechanisms, agent navigation to robotics and games. We select the `Mountain Car Continuous` problem, with the corresponding visual in

[1] https://gymnasium.farama.org/



(a) cRbNMCS at level 2

(b) cNMCS at level 1

Figure 1: Comparison of cRbNMCS and cNMCS on map 15

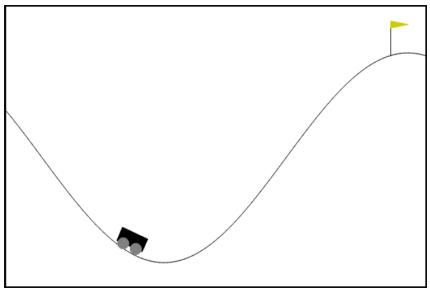


Figure 2: Mountain Car Continuous problem from Gymnasium

figure 2 from Gymnasium's website. It provides continuous observation and action spaces. The goal is to get a car up on a hill to a position $x = 0.45$ using bursts of an engine while the reward penalizes high value actions and only gives a bonus of the goal is reached. An episode terminates when the position reaches that goal, *i.e*, it does not matter if it goes beyond it.

We test test our algorithms and compare them to a selection of baselines which are RL algorithms from the `stable-baselines3`[2] package, mainly a *Proximal Policy Optimization*, a *Advantage Actor Critic*, a *Deep Deterministic Policy Gradient*, and a *Soft Actor-Critic*. For each problem, we only show the best performing baseline in table 5. Algorithms' parameters are displayed in table 4

Parameters	δ	$n_{policies}$	n_{steps}	Baseline's $n_{steps\_learning}$
Values	25	100	10000	100000

Table 4: Parameters of algorithms used for the Mountain Car Continuous problem

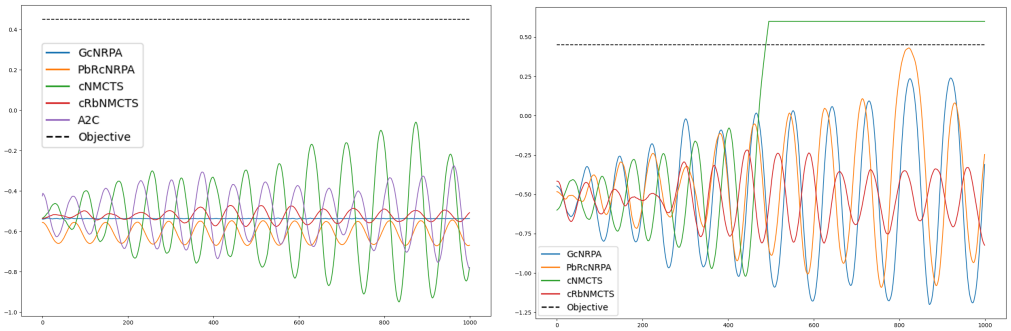
We only run level 1 algorithms because of the value of n_{steps} that would require a tremendous amount of time for level 2. For reference, PbRcNRPA at level 1 takes on average 3 hours and 45 minutes.

When comparing the results in table 5 , we notice that cRbNMCS performs best as it is reward-driven, with GcNRPA also outperforming the baseline. However, we notice that cNMCS is the only algorithm that gets closer to the target than the baseline. In figure 3a, we see that most algorithms, especially GcNRPA and cRbNMCS operate very small movements and cNMCS is the one that tries the widest actions making it the closest to the goal. This is due to the way the reward function is designed, making it hard for the agent to know the objective unless it reaches it, which drives it to reduce the magnitude of its actions and thus stagnate.We then re-run the algorithms with a reward that computes the distance of each position to the goal, truncated to -1 if it is too far, given the shape of the hill on the left side. This version is not compatible with the baselines as they use the built-in reward of

[2]https://stable-baselines3.readthedocs.io/en/master/

Algorithm	cNMCS	cRbNMCS	GcNRPA	PbRcNPRA	Baseline
Best score	-26.8951	**-0.2726**	-0.6236	-17.5918	-13.8399
Shortest distance to goal	**0.510**	0.922	0.985	0.994	0.726

Table 5: Results on Mountain Car Continuous



(a) Environment's reward

(b) Custom reward

Figure 3: Position of the car on the x-axis through time for each algorithm on the Mountain Car problem

the gymnasium environment. We show the new scores in table 6 where the penalty is the previous score, and figure 3b shows the position of the car along the x-axis. We observe that all algorithms do increase the magnitude of their reach, with only cNMCS managing to reach the goal, and in half the time given to other algorithms. The scores however show that large actions are needed for this with cNMCS. Its score is also helped by the fact that it reached the goal and it was boosted by the bonus, whereas cRbNMCS fails even with large actions.

Algorithm	GcNRPA	PbRcNRPA	cNMCS	cRbNMCS
Score	-844.2	-818.8	**-360.9**	-945
Penalty	**-3.32**	-7.29	-32.61	-33.02
Execution time	14156	11724	2155	**403**

Table 6: New scores, penalties and execution times in seconds of the algorithms on the Mountain Car Continuous Problem

4.3 Chemical Processes Control

The last experiments we conduct are on problems from the `pc-gym`[3] package. It contains a set of chemical processes planning problems that aim to reach to final concentration of a product, its absorption rate, its extractions rate, etc. In general, these problems require a well-planned sequence of external actions to such objective [14] and they are bound by differential equations system [16], rendering the choice of actions and their impact on the states rather complicated. The `pc-gym` package offers a simulation environment for a few of these problems, based on the `gymnasium` package.

We compare our algorithms on two problems: the *Continuously Stirred Tank Reactor (CSTR)* and a basic *Nonsmooth Control (NC)*. In each problem, the goal is to choose the actions that make the states' sequence follow a specific sequence that is chosen beforehand, which is called *Setpoints*. At each state, the reward given is the euclidean distance between the observed state and the objective one, and the final score is the cumulative sum of the obtained rewards. We use a baseline, selected in the same way as described in the previous section, and show the results in table 8. Implementation details are highlighted in table 7

For the CSTR problem, we see that at level 1, all algorithms outperform the baseline, especially cRbNMCS. This is explained by its *greedy-like* behavior where it goes where there is the better instant reward. This behavior is suitable for these problems where the objective is to follow a predefined

[3]https://maximilianb2.github.io/pc-gym/

Parameters	δ	$n_{policies}$	Time horizon	n_{steps}	Baseline's $n_{steps\_learning}$
Level 1	20	200	25	100	10000
Level 2	8	50			

Table 7: Parameters of algorithms used for the Chemical Processes problems

Algorithm	cNMCS		cRbNMCS		GcNRPA		PbRcNRPA		Baseline
Level	1	2	1	2	1	2	1	2	
CSTR	0.0232	**0.0052***	0.0084	0.0082	0.0162	0.0098	0.0258	0.0304	0.0548
NC	3.4512	0.3680	**0.1866***	0.2097	2.5503	0.5952	0.9257	0.4026	1.0923

Table 8: Cumulative reward obtained for each algorithm on Chemical Processes.

sequence. It also does achieves this performance much faster than the baseline and NRPA-based algorithms since it does not rely on a policy, and faster than cNMCS as it only evaluates one step instead of a whole trajectory. Increasing the nesting level also increases the execution time by a big margin but only provides a small increase in the overall score, except for cNMCS on the CSTR problem. Indeed, at level 2, it outperforms all other algorithms. To understand this better, we take a look at figure 4. We notice cRbNMCS is very close to the setpoints trajectory which makes it more reliable. Yet cNMCS of level 2 has a lower score on CSTR because at the beginning, it strays the least from the trajectory and follows it relatively well afterwards.

We see a similar behavior for the NC problem, with cRbNMCS at level 1 providing the best results, even more than level 2. This can be explained by the limited bandwidth we chose for level 2, and also helps explain the negligible decrease on the CSTR problem while others show significant improvement when the level increases.

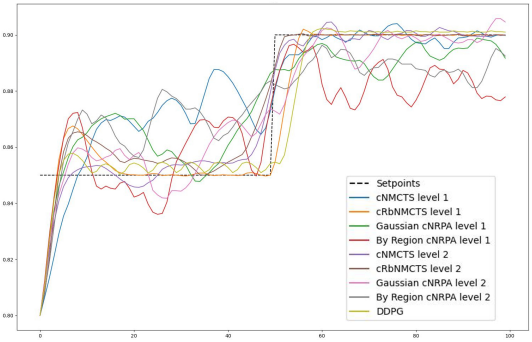


Figure 4: Sequences from different algorithms compared to the setpoints for the CSTR problem

5 Conclusion

Continuous search spaces can be a delicate bottleneck in some real-world instances of operational planning, especially when it comes to classical discrete planning methods. Methods that can operate in this type of spaces are thus more interesting, and leveraging the competitiveness of MCTS-based methods in other applications is indeed relevant. We have shown that our continuous version of NRPA and NMCS display promising results and some of them outperform the chosen baselines.

It does however appear relevant, even mandatory, to further investigate the potential of these methods on other applications and assess their performance against state-of-the-art approaches, especially since they do not rely heavily on memory usage, which makes them rather suitable for embedded systems. For instance, it would be interesting to combine them with a prior as in GNRPA [9] which would help the initial search and reduce the time spent on exploring less promising regions of the search space. Another extension could be to assimilate other statistical features into the computation of regions' values and probability distributions. In essence, more improvements and investigations may provide better insights on the full prowess of Monte Carlo Search when it comes to solving planning problems.

References

[1] H. Akiyama, K. Komiya, and Y. Kotani. Nested monte-carlo search with amaf heuristic. In *2010 International Conference on Technologies and Applications of Artificial Intelligence*, pages 172–176, 2010.

[2] N. Amoussou and D. Pallez. Continuous pareto nested rollout policy adaptation for multiobjective sequential optimization. In *The Genetic and Evolutionary Computation Conference*, San José, Costa Rica, Jul 2026.

[3] A. Antos, C. Szepesvári, and R. Munos. Fitted q-iteration in continuous action-space mdps. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[4] D. Bertsekas. *Reinforcement learning and optimal control*, volume 1. Athena Scientific, 2019.

[5] J. Bresina, R. Dearden, N. Meuleau, S. Ramkrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for ai. *arXiv preprint arXiv:1301.0559*, 2012.

[6] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte. Parametric pomdps for planning in continuous state spaces. *Robotics and Autonomous Systems*, 54(11):887–897, 2006. Planning Under Uncertainty in Robotics.

[7] T. Cazenave. Nested monte-carlo search. In C. Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 456–461, 2009.

[8] T. Cazenave. Generalized rapid action value estimation. In *24th International conference on artificial intelligence*, pages 754–760, 2015.

[9] T. Cazenave. Generalized nested rollout policy adaptation. In *Monte Carlo Search International Workshop*, pages 71–83. Springer, 2020.

[10] G. B. Dantzig. Linear programming. *Operations research*, 50(1):42–47, 2002.

[11] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, 01 2000.

[12] S. Gelly and D. Silver. Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[13] J. Gonzalez and M. Likhachev. Search-based planning with provable suboptimality bounds for continuous state spaces. In *Proceedings of the International Symposium on Combinatorial Search*, volume 2, pages 60–67, 2011.

[14] G. J. Hahn and M. Brandenburg. A sustainable aggregate production planning model for the chemical process industry. *Computers & operations research*, 94:154–168, 2018.

[15] T. Hubert, J. Schrittwieser, I. Antonoglou, M. Barekatain, S. Schmitt, and D. Silver. Learning and planning in complex action spaces. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4476–4486. PMLR, 18–24 Jul 2021.

[16] J. Ingham, I. J. Dunn, E. Heinzle, J. E. Přenosil, and J. B. Snape. *Modelling of Stagewise Processes*, chapter 3, pages 93–172. John Wiley & Sons, Ltd, 2007.

[17] Y. Jia and X. Y. Zhou. Policy gradient and actor-critic learning in continuous time and space: Theory and algorithms. *Journal of Machine Learning Research*, 23(275):1–50, 2022.

[18] B. Kim, K. Lee, S. Lim, L. Kaelbling, and T. Lozano-Perez. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9916–9924, Apr. 2020.

[19] S. Knight, G. Rabideau, S. Chien, B. Engelhardt, and R. Sherwood. Casper: space exploration through continuous planning. *IEEE Intelligent Systems*, 16(5):70–75, 2001.

[20] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[21] Y. Li. Reinforcement learning applications. *arXiv preprint arXiv:1908.06973*, 2019.

[22] M. Lutter, S. Mannor, J. Peters, D. Fox, and A. Garg. Value iteration in continuous actions, states and time. *arXiv preprint arXiv:2105.04682*, 2021.

[23] R. Michelucci, D. Pallez, T. Cazenave, and J.-P. Comet. Improving continuous monte carlo tree search for identifying parameters in hybrid gene regulatory networks. In M. Affenzeller, S. M. Winkler, A. V. Kononova, H. Trautmann, T. Tušar, P. Machado, and T. Bäck, editors, *Parallel Problem Solving from Nature – PPSN XVIII*, pages 319–334, Cham, 2024. Springer Nature Switzerland.

[24] J. Pazis and M. G. Lagoudakis. Reinforcement learning in multidimensional continuous action spaces. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 97–104, 2011.

[25] E. Plaku. Planning in discrete and continuous spaces: From ltl tasks to robot motions. In G. Herrmann, M. Studley, M. Pearson, A. Conn, C. Melhuish, M. Witkowski, J.-H. Kim, and P. Vadakkepat, editors, *Advances in Autonomous Robotics*, pages 331–342, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[26] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.
[27] C. D. Rosin. Nested rollout policy adaptation for monte carlo tree search. In *Ijcai*, volume 2011, pages 649–654, 2011.
[28] M. P. D. Schadd, M. H. M. Winands, H. J. van den Herik, G. M. J. B. Chaslot, and J. W. H. M. Uiterwijk. Single-player monte-carlo tree search. In H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, editors, *Computers and Games*, pages 1–12, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
[29] A. K. Shakya, G. Pillai, and S. Chakrabarty. Reinforcement learning algorithms: A brief survey. *Expert Systems with Applications*, 231:120495, 2023.
[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
[31] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *ICML*, pages 903–910, 2000.
[32] T. Strösslin and W. Gerstner. Reinforcement learning in continuous state and action space. In *Artificial Neural Networks-ICANN 2003*, 2003.
[33] R. S. Sutton. Planning by incremental dynamic programming. In *Machine learning proceedings 1991*, pages 353–357. Elsevier, 1991.
[34] R. S. Sutton, A. G. Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
[35] P. Van Beek and X. Chen. Cplan: A constraint programming approach to planning. In *AAAI/IAAI*, pages 585–590, 1999.
[36] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.
[37] T. Yee, V. Lisỳ, M. H. Bowling, and S. Kambhampati. Monte carlo tree search in continuous action spaces with execution uncertainty. In *IJCAI*, pages 690–697, 2016.

Appendix

To provide more context and statistically significant results that complete the ones shown in the main content, we show here the average score of each algorithm, and its standard deviation.

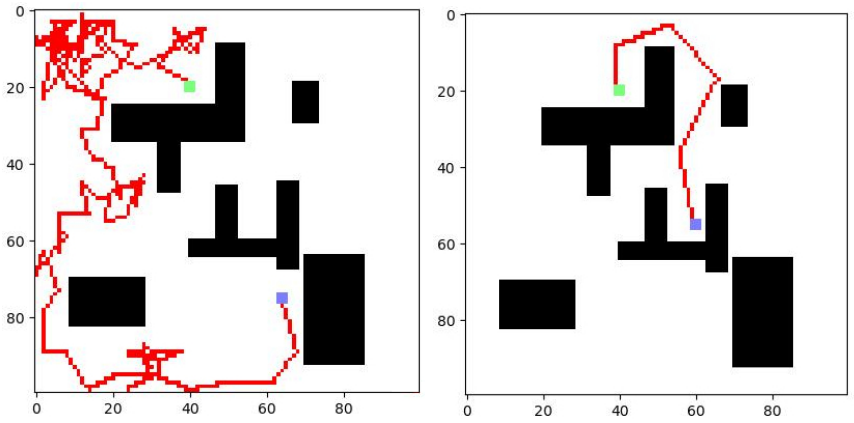
A Map Traveling

Tables 9, 10, and 11 show similar results on average to the minimal scores in terms of which algorithm performs better, with a few exceptions:

- Most algorithms outperform discrete baselines
- Level 2 algorithms perform better than level 1
- cRbNMCS outperforms other algorithms of level 1, except on difficult maps.
- Only cNMCS solves map 15 in level 1.

However, the standard deviation tables 12, 13, and 14 show quite the disparity between algorithms:

- Discrete baselines behave in an almost deterministic fashion
- PbRcNRPA has unstable behavior in level 1, giving a wide range of results
- GcNRPA and cRbNMCS generally provide a low standard deviation, meaning they almost always give the same results.



(a) MCTS on map 11. Score: 1011.0 (b) GcNRPA level 2 on map 12. Score: 65.1

Figure 5: Map Traveling examples

For MCTS, RAVE and GRAVE, the destination is almost reached in most runs, but at the very end of the trajectory, *i.e*, when the maximum size of the trajectory is reached, with an example shown in figure 5a. We also note that GcNRPA navigates rather efficiently on some maps as shown in figure 5b.

We do not discuss execution times here because when algorithms finish the map earlier than the maximum trajectory length, they tend to have shorter execution times, which would make the comparison relatively unfair.

B Mountain Car Continuous

We display statistical results on the Mountain Car Continuous problem in table 15.

C Chemical Processes

We look at results shown in table 16. For the average performance of each algorithm, we observe a similar behavior to the best results, with cRbNMCS providing the best scores at level 1, and doing

Map	1	2	3	4	5
Baselines					
MCTS	1011.0	1011.4	251.0	1011.4	1011.5
RAVE	1012.8	1026.4	794.7	1011.4	1011.8
GRAVE	1012.4	1022.4	794.7	1011.4	1013.1
cMCTS	604.3	406.4	480.3	580.3	376.3
cRAVE	933.3	868.5	980.8	882.2	766.4
cGRAVE	985.6	902.3	948.5	946.4	883.6
Level 1					
cNMCS	762.3	616.3	714.3	703.4	467.3
cRbNMCS	251.3	330.4	267.4	1017.3	256.3
GcNRPA	1017.8	956.8	1001.4	1014.7	891.3
PbRcNRPA	889.4	963.3	873.1	853.7	784.8
Level 2					
cNMCS	416.2	296.2	375.2	423.2	280.2
cRbNMCS	170.3	139.3	146.3	310.8	121.3
GcNRPA	124.7	85.1	114.0	154.3	96.7
PbRcNRPA	274.7	221.5	360.1	364.9	279.0

Table 9: Average scores on maps from 1 to 5 after 10 runs of each algorithm

Map	6	7	8	9	10
Baselines					
MCTS	1013.8	1012.5	1011.0	1011.1	1014.2
RAVE	1013.3	1024.4	1012.6	1014.1	1035.1
GRAVE	1013.5	1022.0	1013.8	1013.5	1034.0
cMCTS	819.3	519.3	530.3	734.5	820.4
cRAVE	1013.2	892.9	938.7	1007.0	1017.6
cGRAVE	990.2	820.8	901.7	952.4	1020.7
Level 1					
cNMCS	676.3	654.3	744.4	818.4	781.8
cRbNMCS	745.8	310.3	262.3	222.4	1061.8
GcNRPA	1021.3	927.7	998.6	1006.7	1024.5
PbRcNRPA	817	607.2	982	812.2	872.7
Level 2					
cNMCS	500.1	344.2	370.2	454.2	445.2
cRbNMCS	201.3	131.4	136.3	152.3	830.0
GcNRPA	159.2	103.3	102.7	113.3	169.1
PbRcNRPA	448.8	230.9	304.3	276.0	653.8

Table 10: Average scores on maps from 6 to 10 after 10 runs of each algorithm

Map	11	12	13	14	15
Baselines					
MCTS	1011.0	1011.0	1013.0	1012.2	1011.0
RAVE	1011	1011.0	1046.5	1012.0	1019.3
GRAVE	1011.0	1011.0	1046.2	1011.7	1017.0
cMCTS	558.4	323.4	908.6	859.6	1014.6
cRAVE	1002.6	755.4	1019.2	987.8	1018.5
cGRAVE	966.6	700.3	1006.7	936.0	1019.7
Level 1					
cNMCS	809.3	590.2	743.4	891.5	997.6
cRbNMCS	1026.5	1026.1	1069.2	1062.7	1045.6
GcNRPA	1011.7	911	1022.7	1016.3	1022.7
PbRcNRPA	930.2	923.7	896.2	1036.1	1033.1
Level 2					
cNMCS	383.2	284.2	537.2	507.1	656.1
cRbNMCS	530.1	346.2	809.3	1015	1030.6
GcNRPA	113.2	101.2	231.0	366.3	548.7
PbRcNRPA	369.7	298.7	481.6	642.1	927.2

Table 11: Average scores on maps from 11 to 15 after 10 runs of each algorithm

Map	1	2	3	4	5
Baselines					
MCTS	0	0.57	37.27	0	0.61
RAVE	1.2	5.24	148.66	0	0.85
GRAVE	1.35	6.42	174.25	0	0.94
cMCTS	114.06	95.72	124.86	127.21	78.74
cRAVE	123.14	174.24	105.59	177.32	157.82
cGRAVE	58.55	156.41	126.06	116.76	188.77
Level 1					
cNMCS	128.6	165.5	95.7	179.4	150.2
cRbNMCS	90.3	126.7	78.2	0.8	169.2
GcNRPA	4.0	111.0	40.4	2.7	192.1
PbRcNRPA	251.3	241.6	287.3	261.1	342.3
Level 2					
cNMCS	44.6	64.7	87	80.2	45.9
cRbNMCS	10	11.4	13.5	251.3	8.3
GcNRPA	16.11	8.21	19.46	20.99	34.65
PbRcNRPA	95.4	126.36	181.39	109.09	122.19

Table 12: Standard deviation of scores on maps from 1 to 5 after 10 runs of each algorithm

Map	6	7	8	9	10
Level 1					
MCTS	2.85	1.17	0	0.37	1.00
RAVE	1.26	5.17	1.55	2.44	7.91
GRAVE	1.55	5.86	1.42	2.16	7.76
cMCTS	121.86	64.2	153.66	183.32	181.82
cRAVE	19.56	107.65	147.73	23.51	6.47
cGRAVE	43.07	212.2	204.16	142.23	4.85
Level 1					
cNMCS	133.7	152.1	190.8	142.3	208.6
cRbNMCS	393.6	181.3	97.8	88.9	2.5
GcNRPA	5.6	180.4	36.8	35.7	8.0
PbRcNRPA	335.0	416.3	114.4	406.1	352.3
Level 2					
cNMCS	63.3	73.9	81.9	72.9	74.7
cRbNMCS	19.2	11.3	19.1	10.8	305.9
GcNRPA	33.68	16.97	17.02	11.56	32.45
PbRcNRPA	245.97	81.04	81.53	136.78	226.66

Table 13: Standard deviation of scores on maps from 5 to 10 after 10 runs of each algorithm

Map	11	12	13	14	15
Baselines					
MCTS	0	0	2.30	1.16	0
RAVE	0	0	6.88	1.29	1.49
GRAVE	0	0	10.98	0.61	3.35
cMCTS	71.79	73.54	112.87	123.76	2.53
cRAVE	20.86	178.65	6.6	86.91	2.93
cGRAVE	87.42	167.06	44.55	143.25	2.7
Level 1					
cNMCS	162.1	167	187.9	141.3	62.4
cRbNMCS	1.5	0.8	0.5	0.7	14.8
GcNRPA	14.1	208.1	5.7	1.3	3.2
PbRcNRPA	241.1	240.3	224.5	12.8	9.8
Level 2					
cNMCS	59.2	42.7	75.6	143	85.8
cRbNMCS	404.9	340.6	388.1	0	4.2
GcNRPA	25.76	23.4	46.18	246.05	205.21
PbRcNRPA	117.34	111.25	231.67	272.71	151.35

Table 14: Standard deviation of scores on maps from 11 to 15 after 10 runs of each algorithm

Algorithm	GcNRPA	PbRcNRPA	cNMCS	cRbNMCS
Average score	-1.3092	-52.0563	-27.6137	-0.6271
Standard deviation	51.49%	39.84%	1.72%	35.03%

Table 15: Average cumulative reward and standard deviation on the Mountain Car Continuous Problem

Problem		CSTR			NC		
Metric		Average	STD	Time	Average	STD	Time
Level 1	cNMCS	0.0373	21.02%	126	5.4258	22.06%	140
	cRbNMCS	0.0088	3.01%	5	0.1970	5.37%	6
	GcNRPA	0.0208	17.22%	913	4.3160	20.64%	634
	PbRcNRPA	0.0807	62.23%	226	24.1437	99.87%	139
Level 2	cNMCS	0.0068	18.69%	13565	0.5178	6.88%	14278
	cRbNMCS	0.0087	5.89%	1051	0.2853	19.68%	745
	GcNRPA	0.0131	10.71%	32583	0.8519	25.96%	30399
	PbRcNRPA	0.0893	88.60%	3982	11.1509	111.61%	4501
Baseline		-	-	3072	-	-	2589

Table 16: Average cumulative reward, standard deviation in percentages and average execution times for each algorithm after 10 runs on each problem. Baseline execution time: sec.

better at level 2 for CSTR. We do observe however different trends for the standard deviation, with cNMCS improving stability when going to level 2 as opposed to cRbNMCS, while PbRcNRPA providing a wide range of results with a high standard deviation at both levels, and GcNRPA providing a stable standard deviation.

For execution times, we do observe that algorithms at level 1 operate faster than the baseline, and only cRbNMCS stays faster at level 2. But we see a disparity, with GcNRPA taking the longest, due to its probability density function computation at each step of the exploration and the adaptation.