ANN-CMCGS: Generalizing Continuous Monte-Carlo Graph Search with Approximate Nearest Neighbors

Christoph Scherer
Technical University of Berlin
Berlin, Germany
c.scherer@tu-berlin.de

Wolfgang Hönig
Technical University of Berlin
Berlin, Germany
hoenig@tu-berlin.de

ABSTRACT

Robot motion planning under uncertainty highlights the need for decision making in continuous domains. Continuous Monte-Carlo Graph Search (CMCGS) meets this need by combining sampling with graph-based reasoning in a unified framework. Existing formulations, however, rely on a layered structure that restricts generality and prevents MCGS from fully realizing its advantages over MCTS. We introduce a non-layered formulation of CMCGS that removes this restriction, enabling a more flexible and scalable search framework applicable to arbitrary directed graphs. To ensure efficiency, we integrate Approximate Nearest Neighbor (ANN) search via Hierarchical Navigable Small-World graphs (HNSW), which allows rapid graph maintenance and querying in high-dimensional continuous spaces. Unlike prior methods that rebuild the graph at every iteration, our approach incrementally updates the graph, bootstrapping the search over time. We demonstrate the benefits in robot motion planning, where non-layered CMCGS with ANN achieves higher success rate compared to layered CMCGS. These results highlight a practical foundation towards real-time, online continuous planning under uncertainty.

KEYWORDS

Monte Carlo Graph Search; Continuous Domains; Motion Planning

1 INTRODUCTION

Online decision-making in continuous domains is a fundamental challenge in robotics, relevant for both closed-loop control and motion planning. In such settings, agents must select actions that achieve long-term objectives under dynamic constraints, often in high-dimensional, uncertain environments.

Robot motion planning traditionally relies on two broad paradigms. Sampling-based motion planners such as PRM [10] or RRT [14, 16] explore the state space via stochastic sampling, enabling global reasoning and asymptotic optimality. While they can be adapted for online planning, they are generally less generic in handling uncertainty and dynamic contexts, and often require problem-specific tuning. Search-based planners, such as A* [7] or CBS [20], systematically explore the state space using heuristic guidance to find feasible, often optimal paths, making them effective for structured environments, but computationally challenging in high-dimensional continuous domains.

Online model-based controllers such as Model Predictive Control (MPC) [6] and Model Predictive Path Integral control (MPPI) [22] generate trajectories over short horizons at each timestep, offering fast reactivity and robustness to disturbances. However, their limited horizon, reliance on dense reward signals, and susceptibility to local minima constrain their applicability in sparse-reward or multi-modal environments. This gap motivates hybrid approaches that combine global reasoning with online adaptability.

Monte Carlo Tree Search (MCTS) [4, 12] offers such a hybrid framework by balancing exploration and exploitation through stochastic sampling. MCTS has been successfully applied to discrete domains and extended to continuous spaces through strategies like progressive widening or state and action abstractions. However, standard MCTS fundamentally assumes a tree structure, which cannot efficiently capture the graph-like connectivity inherent in many continuous planning problems. This leads to redundant exploration and missed opportunities for reusing previously gathered information.

Monte Carlo Graph Search (MCGS) [5, 19] addresses this limitation by incorporating transposition detection, merging equivalent states into directed acyclic graphs. Continuous MCGS (CMCGS) applies this concept to continuous domains but enforces a layered structure, where nodes may only connect to transpositions within the same layer. While simplifying implementation, this restriction limits connectivity and restricts general applicability to high-dimensional motion planning problems where states can be reached via diverse trajectories.

We propose Continuous Monte Carlo Graph Search with Approximate Nearest Neighbors (ANN-CMCGS), a novel formulation for online motion planning that lifts the layered constraint, enabling the construction of arbitrary directed graphs with variable edge lengths and cycles. Our method leverages Approximate Nearest Neighbor (ANN) search via Hierarchical Navigable Small-World (HNSW) graphs [15] for efficient graph construction and transposition detection. Reachability-based checks ensure that transpositions respect the system's dynamics, combining state-space heuristics with controller verification. ANN-CMCGS also incorporates a bootstrapped planning mechanism that incrementally updates the search graph, enabling long-horizon reasoning under sparse rewards.

Our contributions are as follows.

(1) ANN-CMCGS Algorithm: A non-layered Monte Carlo Graph Search formulation that detects approximate transpositions in continuous domains via ANN search, allowing arbitrary directed graphs with cycles and incremental reuse of the search graph. Our results show that
(2) Empirical Evaluation: Demonstrations in robot motion planning showing ANN-CMCGS matches CMCGS in control-oriented settings and substantially outperforms it in higher-dimensional and non-holonomic motion planning problems.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

2 RELATED WORK

Online Planning and Control. Sampling-based planners such as PRM and RRT [10, 14, 16] efficiently explore high-dimensional spaces and reason about feasibility without explicit discretization. However, they have limited generality for dynamic or uncertain environments. Online replanning variants such as Online RRT* and Anytime RRT* [3, 8] improve adaptability but remain constrained by local rewirings and uninformed sampling.

Online model-based controllers such as Model Predictive Control (MPC) [6] and Model Predictive Path Integral control (MPPI) [22] adapt rapidly to disturbances by re-optimizing trajectories at each timestep, but their short horizons and reliance on dense reward signals limit effectiveness in sparse-reward or discontinuous-cost domains. Reinforcement learning approaches offer an alternative that can address some of these limitations, but they often require reward shaping and lack theoretical guarantees and explainability.

Monte Carlo Tree Search in Continuous Domains. Monte Carlo Tree Search (MCTS) is a sampling-based decision-making algorithm that balances exploration and exploitation via stochastic simulations [4, 12]. It has achieved state-of-the-art performance in discrete domains such as board games (e.g., AlphaZero [21]) and combinatorial optimization, thanks to its domain independence, asymptotic optimality, and ability to handle sparse rewards. However, standard MCTS relies on discrete tree structures, which scale poorly in high-dimensional continuous domains and cannot efficiently reuse previously explored states. To address this, extensions such as bandit-based selection strategies like Hierarchical Optimistic Optimization (HOO) [17] or Voronoi Optimistic Optimization (VOO) [11], and state abstraction methods [1] have been proposed to improve efficiency. Other approaches, such as Spectral Expansion Tree Search (SETS) [18], adapt MCTS for real-time robotic planning. Despite these advances, all of these methods still depend on tree structures, which inherently duplicate states and limit the ability to fully exploit the structure of continuous state spaces.

From Trees to Graphs. Extensions of MCTS such as Monte-Carlo Graph Search (MCGS) [5, 19] address the inefficiency of tree structures by merging equivalent states into graphs, enabling transposition detection and avoiding redundant exploration. While exact transpositions are possible in discrete domains such as chess, continuous domains require approximate transpositions based on state similarity. Continuous MCGS (CMCGS) [13] applies this idea using state clustering and Gaussian action bandits within a layered directed acyclic graph, demonstrating feasibility for different control focused robotics tasks including a one dimensional case for motion planning. However, its layered structure restricts connectivity and limits applicability to more general planning tasks. Furthermore, modeling states and actions as Gaussians poses challenges for multi-modal dynamics and introduces implicit assumptions about controllability of the robot. Cyclic graphs for the multi-robot patrolling problem have been explored by Kartal et al. [9], where cycles encode deliberate revisiting behaviors, our use of cycles serves a different purpose. In ANN-CMCGS, cycles arise naturally from possible transitions between states in the search graph, enabling efficient state reuse and long-horizon online planning.

Approximate Nearest Neighbors and HNSW. Nearest neighbor search is fundamental in robotics for identifying similar states during motion planning or collision checking [10, 14]. Exact search, however, becomes intractable in high-dimensional continuous spaces. Approximate Nearest Neighbor (ANN) methods address this by sacrificing minimal accuracy for significant speed and scalability. Among them, Hierarchical Navigable Small-World (HNSW) graphs [15] are particularly effective, organizing data into multi-layer structures that enable fast, incremental queries without rebuilding the index. HNSW has proven efficient in robotic applications such as collision detection [23], and its ability to support dynamic insertion makes it especially well-suited for online graph construction and transposition detection in continuous planning domains.

3 PRELIMINARIES

Markov Decision Process (MDP). We formulate the decision-making problem for motion planning as a Markov Decision Process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P(s'|s,a)$ represents the transition probability from state s to s' given action a, $R(s,a)$ is the reward function, and $\gamma \in [0,1]$ is the discount factor. A policy $\pi(a|s)$ maps states to probability distributions over actions, and the objective is to find an optimal policy π^* that maximizes the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right].$$

The value function $V^\pi(s)$ and the action-value function $Q^\pi(s,a)$ quantify the expected return under policy π from state s or state-action pair (s,a), respectively.

Monte Carlo Tree Search (MCTS). MCTS is a simulation-based planning algorithm that incrementally builds a search tree by balancing exploration and exploitation. It is particularly suited for problems where explicit models of the state space are unavailable or expensive to enumerate. MCTS iteratively executes four phases:

(1) **Selection:** Starting from the root, recursively select child nodes according to a selection policy such as Upper Confidence bounds applied to Trees (UCT) until a leaf node is reached.
(2) **Expansion:** If the leaf node is not terminal, expand the tree by adding one or more child nodes corresponding to possible actions.
(3) **Simulation:** From the newly added node, perform a rollout (simulation) to estimate the expected return, usually using a default policy.
(4) **Backpropagation:** Propagate the obtained return back through the visited nodes to update their value estimates.

The UCT selection rule is given by:

$$a^* = \arg\max_{a} \left[Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}}\right],$$

where $Q(s,a)$ is the mean action value, $N(s)$ is the visitation count of state s, $N(s,a)$ is the visitation count for (s,a), and c is a constant controlling exploration [12]. For finite-horizon problems, UCT in particular has been shown to converge to an optimal solution at a polynomial rate [12].

Monte Carlo Graph Search (MCGS). To extend the tree search to graph search, Saffidine et al. [19] and Czech et al. [5] introduced mechanisms for extending the tree to a directed acyclic graph (DAG) and improving the efficiency of MCGS. One approach is to perform transposition checks during node expansion, where generated states are hashed and compared against previously encountered states. If a transposition has been found, instead of inserting the candidate state to a new node, the node chosen for expansion is connected to the existing transposition node in the graph by an edge representing the chosen action. Although value updates along all possible ancestor nodes and trajectories have been discussed in the literature, the standard formulation performs backpropagation only along the trajectory traversed during the selection phase [5]. Another contribution is the introduction of ϵ-greedy selection [5], which introduces stochasticity into the action-selection process to balance exploration and exploitation.

State-Space Metric Heuristic. A key requirement for both clustering methods and approximate transposition detection using nearest-neighbor search is the definition of a suitable state-space distance metric. In our formulation, this metric serves as a heuristic for the ANN search and does not need to perfectly capture system dynamics, since all candidate transpositions are subsequently verified through the controller-based reachability check. Allowing mild overestimation of similarity is acceptable, as it merely increases the number of candidates passed to the controller for verification. Because ANN methods such as HNSW inherently provide only approximate recall, the metric and search radius combined determine the trade-off between computational cost and the likelihood of detecting all relevant transpositions. The metric therefore is model-specific but can remain simple, chosen to reflect the dominant geometric or kinematic relations within the state representation.

4 CONTINUOUS MONTE CARLO GRAPH SEARCH WITH APPROXIMATE NEAREST NEIGHBORS

Our proposed method, **Continuous Monte Carlo Graph Search with Approximate Nearest Neighbors (ANN-CMCGS)**, extends Monte Carlo Graph Search (MCGS) to support online planning in continuous domains. The key idea is to replace exact transposition checks with approximate nearest neighbor (ANN) search, enabling scalable graph maintenance and real-time updates. Unlike previous layered formulations of MCGS, our approach constructs non-layered graphs with arbitrary connectivity and edge lengths, allowing cycles and richer topologies that better represent motion planning problems.

ANN-CMCGS follows the standard Monte Carlo search paradigm but generalizes it from a tree to an arbitrary directed graph. Each iteration consists of node selection, expansion, simulation, and backpropagation, adapted to handle continuous state–action spaces and approximate transpositions. This formulation supports online operation in a closed-loop fashion: after each environment step, the current state is either mapped to an existing node or inserted as a new root node into the existing search graph via the ANN index, allowing the planner to reuse past computations and maintain long-horizon consistency across planning episodes.

For approximate transposition detection we leverage the concept of reachability for a rigorous and intuitive way of implicitly clustering states, so for any given node, all children belong to its reachable set. For that we explicitly assume to have access to a controller that we can query to check for reachability between any two given states. Combined with a heuristic state space metric that approximates the proximity of nodes by reachability required by the nearest-neighbor search, we can find approximate transpositions that are similar to the candidate state and ensure that it is reachable using the controller. This two-stage process is necessary, as finding a suitable state space metric in high-dimensional spaces is difficult, so that classical clustering methods become infeasible. This assumption is not strict, as the problem is inherent and other approaches like [13] that use clustering are susceptible to it. In practice, provided a sufficiently accurate state space metric exists that allow for radius search in the ANN query, querying the controller to check for reachability becomes unnecessary. For simple dynamics such as single integrator dynamics such a metric is trivial, while suitable metrics for more complex dynamics could be derived.

When connecting a node to an approximate transposition, the action that was used to propagate the candidate state becomes inaccurate. One way to approach this issue is to approximate the action distribution using Gaussians as proposed in [13]. However, for depending on system dynamics the transitions may not be described by a single Gaussian distribution within a node well. In this case the action representations can lead to problems during execution as well, arising the need for a controller anyway. In our approach, each edge created is ensured to be executable without restriction on the dynamics.

In summary, the main algorithmic extensions over standard MCGS are:

(1) **Non-layered graph representation** that supports cycles and variable edge lengths.
(2) **Acyclic node selection** in arbitrary directed graphs.
(3) **Online planning with incremental graph updates** across environment steps.
(4) **Heuristic diversity-based expansion** to maximize information gain and minimize redundant transpositions.
(5) **Approximate transposition detection** via ANN search and local controller verification.
(6) **Backpropagation** from parallel expansions.

Graph Representation and State Management. The search structure in ANN-CMCGS is a directed graph $G = (V, E)$, where each node $v = \langle s, N, Q, U, F, T, \bar{s} \rangle \in V$ corresponds to a tuple that includes state $s \in \mathcal{S}$, node visit count N, Q-values Q, utility values from the rollout U, flags for successful termination F or truncation T as well as an encoding of a the state $\bar{s}$ that is used for the ANN lookup and action sampling. Each edge $e = \langle v_i, v_j, a, N \rangle \in E$ represents a feasible transition between states, generated by simulating an action $a \in \mathcal{A}$ from $v_i \in V$. The edge stores the corresponding action sequence a, and edge visit count N used for value estimation and selection. Storing the Q-values in the nodes and not in the edges allows for value updates to spread faster. Storing in the edges would update the statistics only along the direct edge in the playout path, imposing a strong imbalance between edge Q-values.

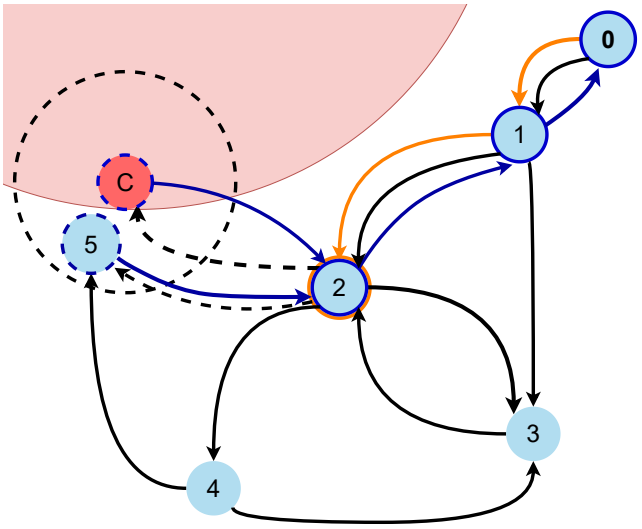

Figure 1: Illustration of graph expansion in ANN-CMCGS through four sequential steps. Black arrows indicate existing edges, while orange and blue arrows mark node selection and backpropagation. (1) Starting from the root node 0, node 2 is selected for expansion. (2) From 2, a candidate node C is sampled; red denotes truncation due to collision with an obstacle. (3) The ANN insertion finds node 5 as a reachable and non-truncated transposition within the radius around C and adds both 5 and C and the corresponding actions as new children (dashed). (4) During backpropagation, C, 5, its edges and all nodes and edges along the playout path are updated toward the root (blue).

Additionally, the index of each node and its encoded state are stored in the HNSW index.

In contrast to tree-based MCTS or layered CMCGS, our formulation allows arbitrary connections between nodes, enabling cycles and heterogeneous edge lengths. This flexibility is essential in motion planning, where positions are typically reachable by a multitude of trajectories. Finally, by allowing variable-length edges, ANN-CMCGS naturally supports integration with motion primitives or learned local controllers. Each edge can represent a short-horizon action sequence, enabling a unified treatment of discrete and continuous dynamics within the same graph framework which is required for motion planning, as many robot dynamics do not allow for exact tracking of a state within a single action. Adding every intermediate step as a node would cause combinatorial growth, so each node instead summarizes the relevant portion of its reachable set.

Online Planning Loop. During the episode the algorithm performs four basic steps. The graph is expanded as long as computational budget remains. After the budget has been depleted, the best action is retrieved. After each decision step, the newly observed state is treated the same as the candidate node that is inserted using the ANN insertion function with $k = 1$ described in the following paragraphs to find the maximum of one transposition. The returned node becomes the new root node for planning in the next step. If the new root node is not a node already in the graph, the planning algorithm will reconnect the graph during the next iteration provided the agent is in a state where further expansion is still possible. If there is no valid action available, a random action is executed instead. This mechanism allows the planner to bootstrap from previous computations, preserving long-horizon knowledge while adapting to real-world deviations in execution.

The overall online planning loop combining these components is summarized in Algorithm 1.

Algorithm 1 ANN-CMCGS — Online Planning Loop

procedure ANN-CMCGS(s_0, B, k, ϵ)
  Initialize graph G with root node $v_0 \leftarrow s_0$
  Initialize ANN index with v_0
  while task not complete **do**
    for $i = 1$ to B **do**
      $v, path \leftarrow$ SelectNode(v_0, ϵ) ▷ See "Node selection"
      ExpandGraph($v, path$)
    end for
    $a^* \leftarrow$ SelectBestAction(v_0)
    $(s', v') \leftarrow$ EnvironmentStep(a^*)
    $v_0 \leftarrow$ InsertWithANN(v', k) ▷ See "Insertion using ANN"
    CollectGarbage
  end while
end procedure

Node Selection. Starting from the root, the algorithm recursively selects successor nodes according to the UCT criterion. To control the effective width of the search tree in continuous action spaces, we introduce a stochastic stopping criterion for tree descent. At each iteration during node selection, the descent terminates with probability ϵ, in which case the currently selected node is expanded; otherwise, with probability $1 - \epsilon$, UCT-based selection continues to a successor node. During UCT selection, only non-truncated successors are considered, and nodes already contained in the current playout path are excluded to prevent cyclic traversals. This ensures that even though the underlying graph may contain cycles, each playout remains acyclic.

Efficient Node Sampling. To maximize exploration efficiency during node expansion, a fixed number D of candidate actions $a_i \sim \mathcal{A}$ are sampled from the continuous action space of the selected node. Each action is simulated to yield a corresponding successor state $s'_i = f(s, a_i)$. From these, the candidate that maximizes its minimum distance to all existing children of the node is chosen, thereby promoting dispersion in the reachable set:

$$(a^*, s^*) = \arg\max_{(a_i, s'_i)} \min_{v_j \in \text{Children}(v)} d(s'_i, s_j),$$

where $d(\cdot, \cdot)$ is the heuristic state-space metric also used in the ANN index. This diversity-based sampling favors actions leading to previously unexplored regions of the state space, reducing redundant transposition checks and improving coverage efficiency. The initial node utility is evaluated using the reward function from a random rollout. The rollout performs a sequence of random actions of fixed length and returns the reached state.

Insertion using ANN. For the sampled candidate state, an ANN query is performed to detect approximate transpositions within a defined similarity radius, using the heuristic state-space metric. A consequence of the radius-based search is the possibility of finding more than one transposition at a time. This promotes the high connectivity in the graph further and allows for faster convergence of Q-values. Since our implementation relies on *HNSWlib*[1], which does support online insertion of new elements important for the iterative graph expansion, it does not allow for a radius search natively. As a consequence we perform the radius check after querying for a large number of k neighbors which the implementation supports. On the nodes returned by the query, the controller-based check is performed to further filter out already connected transpositions and to ensure actual reachability as well as create the edges that can connect the given transposition. The edges found are inserted in the graph. The candidate node itself is only inserted in the graph under the condition that either no transposition was found, or no new edge that has been created lead to a node with the same termination or truncation flag: A critical node like one that truncated because of an obstacle collision should be included if it carries new information that is not yet available by the transpositions detected. This is visualized in Figure 1. The HNSW index is updated accordingly if a new node is inserted in the graph. This expansion mechanism balances computational efficiency with robust graph coverage.

Backpropagation. As with every iteration more than one new edge can be created at a time, both the edge and the associated target node are updated according to the update rules outlined in Algorithm 2. After that, the remaining backpropagation is performed along the playout path in reverse order until the root node has been reached. A related concern to the possible cycles in the node selection arises during backpropagation, where Q-values are mutually dependent. However, because playout paths are cycle-free, each backpropagation step remains finite. This problem also has major implications for the convergence of Q-values discussed in the limitations section.

Garbage Collection. In some domains, such as space–time planning, the search graph may have an inherent temporal structure where actions can only progress forward in time. As a result, nodes may eventually become impossible to reconnect, for example, when their associated timestamps are in the past. During ANN lookup, these outdated or unreachable nodes can accumulate and degrade computational efficiency. To address this, a periodic garbage-collection strategy can remove nodes based on their lifetime or reachability, keeping the graph compact and focused on the relevant regions of the state space. Although this process may temporarily disconnect certain nodes or subgraphs, they can be naturally reconnected during subsequent graph expansions.

Algorithm 2 Graph expansion

procedure ExpandGraph($v, path$)
  $v_c, a \leftarrow$ SampleNode(v) ▷ See "Efficient Node Sampling"
  $V'_{new} \leftarrow$ InsertWithANN(v, v_c, a, k) ▷ See "Insertion using ANN"
  for $v'_{new} \in V'_{new}$ **do**
    UpdateNode(v'_{new})
    UpdateEdge(v, v'_{new})
  end for
  Backpropagate($v, path$)
end procedure
procedure UpdateNode(v)
  $(E_v, V'_v) \leftarrow$ outgoing edges and child nodes from v
  $N(v) \leftarrow 1 + \sum_{e \in E_v} N(e)$
  $Q(v) \leftarrow \frac{1}{N(v)} (U(v) + \sum_{(e,v') \in (E_v, V'_v)} N(e) \cdot Q(v'))$
end procedure
procedure UpdateEdge(v, v')
  $e \leftarrow$ edge connecting v to v'
  $N(e) \leftarrow 1 + N(e)$
end procedure
procedure Backpropagate($v, path$)
  if $path \neq \emptyset$ **then**
    $v_{parent} \leftarrow path$.pop()
    UpdateEdge(v_{parent}, v)
    Backpropagate($v_{parent}, path$)
  end if
end procedure

5 EXPERIMENTS

The Continuous Monte Carlo Graph Search (CMCGS) algorithm was originally proposed for closed-loop control in continuous domains, where online decision-making proceeds incrementally through repeated replanning. In this work, we show that while CMCGS performs well in control domains, it struggles in more exploration heavy online planning tasks like robotic motion planning.

Our experimental evaluation therefore focuses on motion planning problems, not to benchmark motion planning itself, but as one way to test the generality and robustness of CMCGS and our proposed ANN-CMCGS extension, which requires more exploration on longer planning horizons. These environments expose structural limitations of layered graph search such as its inability to leverage cyclic reachability, providing a controlled setting for comparing both methods under identical computational budgets. For this, we focus on planning in space (rather than space-time) for all new benchmarks proposed, with one additional benchmark from the CMCGS baseline that has inherent temporal structure and therefore does not allow cycles.

We focus our comparison on CMCGS as it has already been shown to outperform alternative Monte Carlo and sampling-based planning approaches (e.g., MCTS, VOOT, CEM) by a wide margin in continuous control and therefore serves as the strongest available baseline. Our goal is to demonstrate that ANN-CMCGS preserves the efficiency of CMCGS in its original regime while extending its applicability to richer online planning domains where the baseline fails. Our goal is to demonstrate that ANN-CMCGS retains the efficiency and scalability of CMCGS in its intended regime, while additionally handling the richer dynamics of motion planning tasks that render the baseline inapplicable.

[1] https://github.com/nmslib/hnswlib

5.1 2D Navigation Environment

We adopt the 2D Navigation environment from the CMCGS paper by Kujanpaa et al. [13]. In its original form, the dynamics represent those of a 1D double integrator, since the robot advances along the x-axis with fixed velocity and only needs to regulate its lateral position. As this poses more a continuous control problem, where the controls have to be very precise to solve the problem, it requires fine control and exploration in the action space in its only dimension, making it easy to solve for both CMCGS and MCTS variants that do not leverage transpositions. This is substantially different to the higher dimension problems that can actually leverage high connectivity from transpositions like apparent in motion-planning problems. In addition, the fixed velocity along the x-axis imposes the temporal structure, making cycles impossible as there is an explicit relation of the robot-environment interactions with the current environment step. We extend this environment with progressively more complex robot dynamics: 2D single integrator, 2D double integrator and single integrator unicycle. All benchmarks use a similar sparse reward structure to isolate search efficiency from task-specific design: Rewards give a positive signal only when the environment returns success for a given state. However, for the original benchmark, we did not change the reward structure to ensure consistency comparing to the baseline method. For this, a small reward was given for any environment step that does not truncate. Since we use the same hyperparameters that were proposed in the published work, tweaking the reward structure too much could impact the performance of the baseline method, requiring adjustments to the hyperparameters. However, the original benchmark is ensured to terminate after nine steps or truncate after ten steps. In the scenarios we propose, the agents could traverse the environment in infinite steps or until the episode is terminated. Giving a reward for every step not truncating would therefore promote behavior like going in circles, artificially increasing rewards without making actual progress. Additionally, the baseline rewards function gave a slight penalty for stronger control inputs. While this makes sense for controller design where a controller is often supposed to minimize control input either for stability, time saving, or for energy efficiency purposes, here we focus on the general ability to find feasible solutions within as little steps as possible, which would dominate the effect of conserving time and energy.

The reward function for any given step therefore is given by

$$R(s) = 0.01 \cdot \neg\text{trunc}_{1D}(s) + 0.97 \cdot \text{term}(s) - 1 \cdot \text{trunc}(s) \quad (1)$$

where $\text{term}(s)$, denotes the environment's return of a successful termination flag and $\text{trunc}(s)$ a truncation flag respectively. $\text{trunc}_{1D}(s)$ denotes truncation if the current environment uses 1D double integrator dynamics, mirroring the reward of the original benchmark as closely as possible. Each of these functions returns $r \in \{0, 1\}$ depending on if the flag returned is *True* or *False*.

5.2 Robot Dynamics

The dynamics for the single integrator, double integrator, and unicycle models follow standard formulations. The dimensionality of their respective state and action spaces is shown in Table 1. The increasing nonlinearity and coupling in these dynamics lead to more challenging exploration and highlight the scalability limits of methods relying on strictly layered graph representations.

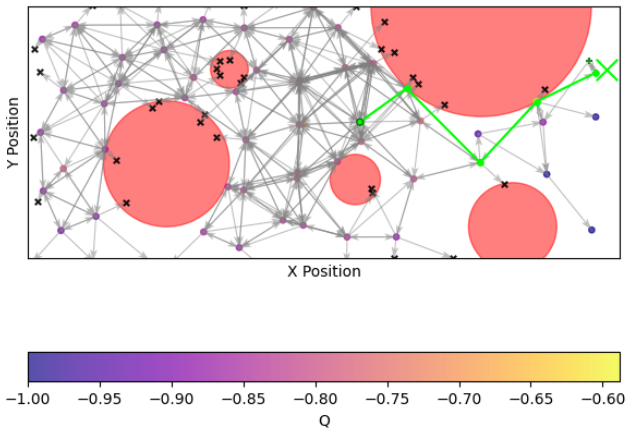

Figure 2: Visualization of the navigation graph in configuration space built during execution using 2D single integrator dynamics starting from the left towards the goal denoted by the large green cross. The green circle with the black outline indicate the current position of the agent and its planned trajectory for the next steps. The color of the remaining nodes indicate their current Q-value. Nodes denoted by a black cross indicate a truncation, whereas the green node close to the goal denoted by "+" indicate a termination.

Table 1: Benchmark environments and robot properties.

Environment	State Dim.	Action Dim.
1D Double Integrator	2	1
2D Single Integrator	2	2
2D Double Integrator	4	2
Unicycle	3	2

5.3 Computational Budgets and Online Execution

All methods compared operate under identical computational budgets for each environment step. After every planning cycle, the best action returned by the planner is executed, and the resulting new state is inserted into the existing search graph using the ANN index. This procedure allows to reuse previous computations across time steps, supporting a closed-loop online planning setting under realistic computational constraints.

5.4 Optimal Control with CasADi

To verify transpositions and ensure local dynamic feasibility, we use a simple finite-horizon optimal control formulation implemented in CasADi [2]. Given an initial state s_0 and a target s^*, the controller optimizes a sequence of control inputs $a_{0:H-1}$ over a fixed horizon H, matching the rollout length used during graph expansion. The cost function penalizes deviation from the target state while respecting system dynamics. CasADi's symbolic differentiation and

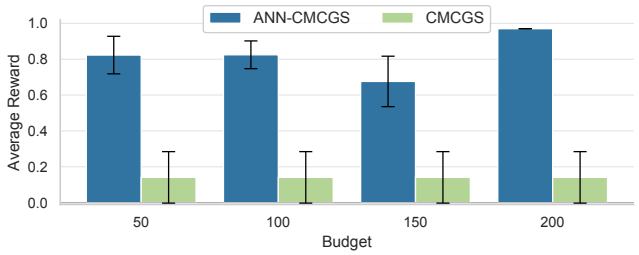

Figure 3: Performance of ANN-CMCGS (ours) and CMCGS with unicycle dynamics across increasing node expansion budgets. Results averaged over 20 episodes; error bars show standard error of mean.

IPOPT solver are used to compute the control sequence. While this is not necessarily the most efficient approach for any given dynamic system and therefore renders experiments on wall clock time less meaningful, this consistent setup ensures fair and comparable reachability checks across all dynamics models covered here. In practice, computationally efficient specialized controllers would be used to overcome the bottleneck of solving optimal control problems.

5.5 Hyperparameter Tuning

Hyperparameters such as exploration coefficients, rollout length, and controller weights strongly influence planner performance and are sensitive to both the system dynamics and the reward structure. To ensure a fair comparison between methods, we applied Bayesian optimization to tune the hyperparameters of the CMCGS baseline on each benchmark except the one originally covered. The optimization maximized the final episode return over a fixed evaluation budget using Gaussian process regression with an expected-improvement acquisition function. For ANN-CMCGS, we adopted the same hyperparameters across all benchmarks with only minimal manual adjustment, emphasizing robustness over problem-specific tuning. This setup deliberately favors the baseline by allowing automated optimization while keeping our method mostly hand-tuned. The resulting configuration provided a balanced trade-off between stability and exploration efficiency, demonstrating that ANN-CMCGS performs competitively without extensive hyperparameter search.

5.6 Results

2D Navigation Environment . The results across all benchmark environments are summarized in Table 2. As expected, both CMCGS and ANN-CMCGS solve the original 1D single-integrator task with identical performance, confirming that the proposed modification preserves the strong efficiency and convergence properties of the baseline in control-oriented settings. However, even MCTS (using the same parameters as ANN-CMCGS with disabling transposition detection by setting the radius for nearest-neighbor search to zero) solves the benchmark with a high success rate, confirming that the benchmark is not suitable to fully show the advantage of transposition usage.

However, performance differences become pronounced as the dimensionality and dynamic coupling of the environment increase. In the 2D single-integrator and double-integrator benchmarks, CMCGS exhibits a sharp degradation in both success rate and average return, frequently failing to reach the goal within the allocated planning budget. In contrast, ANN-CMCGS maintains substantially higher success rates and positive average returns, highlighting its improved ability to explore and reuse previously discovered states.

The trend continues in the unicycle environment, where nonlinear dynamics and cyclic reachability pose significant challenges to strictly layered search. Here, ANN-CMCGS achieves nearly three times the success rate of CMCGS under identical computational constraints.

Budget Analysis. As shown in Figure 3, ANN-CMCGS achieves high rewards on average with as little as 50 node expansions per step with increasing computational budget having little effect, although a budget of 200 node expansions per step led to a success rate of 100% achieving the maximum reward of 0.97 . In contrast, CMCGS shows no improvement even when the budget is increased fourfold, indicating that additional computation does not translate into better planning or higher success rates in this setting. This highlights a fundamental difference in how the two methods scale with budget and their ability to leverage extra resources for reliable performance.

Overall, our experiments suggest that while CMCGS performs well in settings aligned with its original design for closed-loop control, its performance degrades in exploration-heavy online planning tasks with possible cycles. ANN-CMCGS consistently outperforms the CMCGS baseline under these conditions, demonstrating that the additional flexibility from handling cycles and iterative graph expansion enables more effective reuse of previously explored states. However, even with increased computational budgets, the performance gap remains bounded particularly in the most complex benchmarks. This indicates that the proposed tasks are sufficiently challenging to reveal nuanced differences between methods that simpler benchmarks, such as the original 1D double integrator dynamics, fail to expose.

In summary, these results suggest that ANN-CMCGS preserves the efficiency of the baseline in control-oriented tasks while extending its applicability to richer, more exploration-driven planning problems. This highlights its potential as a more general tool for continuous decision-making without sacrificing computational efficiency.

6 LIMITATIONS

Although ANN-CMCGS shows strong performance and scalability, several limitations remain. Performance wise, the most critical bottleneck is the controller query, which can dominate runtime during transposition verification. Future work could should learning based methods for reachability checks, reducing the number of expensive controller calls. Alternatively, more efficient controller algorithms could be derived for specific dynamics models to improve computational efficiency.

Another open challenge is node selection. Ideally, the graph should expand preferentially at the boundaries of the explored region rather than inside already well-covered areas. The current selection criteria only ensure that all nodes are selected infinitely

Table 2: Comparison across all benchmarks. Each benchmark lists results using the same configuration parameters (computational budget per step indicated, where the budget is the number of nodes expanded during each step). Each benchmark was performed over 15 episodes.

Benchmark	Method	Avg. Reward (↑)	Std. Reward	Avg. Steps (↓)	Success % (↑)
1D Single Integrator (50)	ANN-CMCGS	**1.060**	0.000	**9.0**	**100.0**
	CMCGS	**1.060**	0.000	**9.0**	**100.0**
	MCTS	0.929	0.491	9.1	93.3
2D Single Integrator (50)	ANN-CMCGS	**0.776**	0.388	26.7	**80.0**
	CMCGS	-0.475	0.871	**12.7**	26.7
2D Double Integrator (100)	ANN-CMCGS	**-0.145**	0.943	21.3	**40.0**
	CMCGS	-1.000	0.000	**2.8**	0.0
Unicycle (50)	ANN-CMCGS	**0.841**	0.330	**26.7**	**86.7**
	CMCGS	0.059	0.669	41.1	26.7

often and can lead to redundant local refinements. Adaptive exploration strategies that explicitly bias expansion toward frontier nodes may further improve sample efficiency.

Finally, the theoretical properties of ANN-CMCGS require further study. While tree/DAG-based methods ensure that statistics propagate monotonically toward the root, different playout paths in cyclic graphs may induce updates in opposing directions. This complicates the theoretical analysis of convergence and completeness, which we do not address in this work. Empirically, however, we did not observe behaviors such as oscillatory expansion or repeated switching between nodes. We partly attribute this to the stochastic progressive widening approach used.

Additionally, the interaction between ANN recall errors, controller quality, and the stochastic exploration process remains an open area for rigorous analysis.

7 FUTURE WORK

Future work will broaden the evaluation of the theoretical properties of ANN-CMCGS and extend its algorithmic capabilities. This includes testing on diverse continuous-control benchmarks to assess generalization across system dynamics and reward structures, clarifying the boundary between control-oriented and exploration-heavy domains.

The ANN framework could be further leveraged beyond transposition detection, such as accelerating collision checking via ANN-based obstacle maps, unifying search, reachability verification, and safety within a scalable structure. Extending ANN-CMCGS to multi-agent domains is another promising direction, enabling efficient coordination through approximate transposition and reachability checks in joint or factored state spaces.

Finally, integrating learning-based approaches could enhance transposition detection and enable end-to-end optimization of exploration and evaluation within the ANN-CMCGS framework.

8 CONCLUSION

We introduced ANN-CMCGS, a generalization of Continuous Monte Carlo Graph Search that uses approximate nearest-neighbor search and a reachability-based transposition paradigm. The method supports sparse rewards and continuous dynamics, efficiently reuses computational resources across planning steps, and extends to arbitrary graph structures without being restricted to a specific task domain, provided a local controller exists. By leveraging reachability for transposition detection, it avoids strong assumptions about the system model while remaining practical for high-dimensional problems.

Across all introduced benchmarks, ANN-CMCGS achieves significantly higher success rates and computational efficiency than the original CMCGS baseline, solving instances where the baseline fails. These results suggest that approximate graph-based search, combined with controller-informed reachability, provides a powerful foundation for online planning in continuous domains, bridging the gap between control and motion planning.

ACKNOWLEDGMENTS

This work has been supported by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG).

REFERENCES

[1] David Abel, David Hershkowitz, and Michael Littman. 2016. Near Optimal Behavior via Approximate State Abstraction. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, New York, New York, USA, 2915–2923. https://proceedings.mlr.press/v48/abel16.html

[2] Joel A. E. Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. 2019. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation* 11, 1 (2019), 1–36. https://doi.org/10.1007/s12532-018-0139-4

[3] Bryant Chandler and Michael A. Goodrich. 2017. Online RRT* and online FMT*: Rapid replanning with dynamic cost. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vancouver, BC, Canada, 2017-09). IEEE, 6313–6318. https://doi.org/10.1109/IROS.2017.8206535

[4] Rémi Coulom. 2006. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proceedings of International Conference on Computers and Games (ICCG)*. Springer, Berlin, Heidelberg, 72–83.

[5] Johannes Czech, Patrick Korus, and Kristian Kersting. 2021. Improving AlphaZero Using Monte-Carlo Graph Search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, Vol. 31. AAAI Press, 103–111. https://doi.org/10.1609/icaps.v31i1.15952

[6] Carlos E. García, David M. Prett, and Manfred Morari. 1989. Model predictive control: Theory and practice—A survey. *Automatica* 25, 3 (1989), 335–348. https://doi.org/10.1016/0005-1098(89)90002-2

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[8] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. 2011. Anytime Motion Planning using the RRT*. In *2011 IEEE International*

Conference on Robotics and Automation (ICRA). 1478–1483. https://doi.org/10.1109/ICRA.2011.5980479

[9] Bilal Kartal, Julio Godoy, Ioannis Karamouzas, and Stephen J. Guy. 2015. Stochastic Tree Search with Useful Cycles for patrolling problems. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015-05). 1289–1294. https://doi.org/10.1109/ICRA.2015.7139357

[10] Lydia E Kavraki, Petr Svestka, Jean Latombe Latombe, and Mark H Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12, 4 (1996), 566–580. https://doi.org/10.1109/70.508439

[11] Beomjoon Kim, Kyungjae Lee, Sungbin Lim, Leslie Kaelbling, and Tomas Lozano-Perez. 2020. Monte Carlo Tree Search in Continuous Spaces Using Voronoi Optimistic Optimization with Regret Bounds. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 6 (2020), 9916–9924. https://doi.org/10.1609/aaai.v34i06.6546

[12] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Proceedings of the European Conference on Machine Learning (ECML)* (Berlin, Germany). Springer, Berlin, Heidelberg, 282–293. https://doi.org/10.1007/11871842_29

[13] Kalle Kujanpää, Amin Babadi, Yi Zhao, Juho Kannala, Alexander Ilin, and Joni Pajarinen. 2024. Continuous Monte Carlo Graph Search. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Auckland, New Zealand). International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 1047–1056.

[14] Steven Lavalle and James Kuffner. 2000. Rapidly-Exploring Random Trees: Progress and Prospects. *Algorithmic and computational robotics: New directions* (01 2000), 303–307.

[15] Yu A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 42, 04 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[16] Dena Kadhim Muhsen, Firas Abdulrazzaq Raheem, and Ahmed T. Sadiq. 2024. A Systematic Review of Rapidly Exploring Random Tree RRT Algorithm for Single and Multiple Robots. *Cybernetics and Information Technologies* 24, 3 (sep 2024), 78–101. https://doi.org/10.2478/cait-2024-0026

[17] Rémi Munos. 2014. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. *Foundations and Trends® in Machine Learning* 7, 1 (2014), 1–129. https://doi.org/10.1561/2200000038

[18] Benjamin Rivière, John Lathrop, and Soon-Jo Chung. 2024. Monte Carlo tree search with spectral expansion for planning with dynamical systems. *Science Robotics* 9, 97 (2024). https://doi.org/10.1126/scirobotics.ado1010

[19] Abdallah Saffidine, Tristan Cazenave, and Jean Méhat. 2012. UCD : Upper confidence bound for rooted directed acyclic graphs. *Knowledge-Based Systems* 34 (2012), 26–33. https://doi.org/10.1016/j.knosys.2011.11.014 A Special Issue on Artificial Intelligence in Computer Games (AICG).

[20] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. 219 (2015), 40–66. https://doi.org/10.1016/j.artint.2014.11.006

[21] 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. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404

[22] Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. 2017. Model Predictive Path Integral Control: From Theory to Parallel Computation. *Journal of Guidance, Control, and Dynamics* 40, 2 (2017), 344–357. https://doi.org/10.2514/1.G001921

[23] Xiaofeng Zhang, Bo Tao, Du Jiang, Baojia Chen, Dalai Tang, and Xin Liu. 2024. Novel Probabilistic Collision Detection for Manipulator Motion Planning Using HNSW. *Machines* 12, 5 (2024), 321. https://doi.org/10.3390/machines12050321