Learning Space Partitions for Path Planning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Path planning, the problem of efficiently discovering high-reward trajectories, often requires optimizing a high-dimensional and multimodal reward function. Popular approaches like CEM [37] and CMA-ES [16] greedily focus on promising regions of the search space and may get trapped in local maxima. DOO [31] and VOOT [22] balance exploration and exploitation, but use space partitioning strategies independent of the reward function to be optimized. Recently, LaMCTS [44] empirically learns to partition the search space in a reward-sensitive manner for black-box optimization. In this paper, we develop a novel formal regret analysis for when and why such an adaptive region partitioning scheme works. We also propose a new path planning method `PlaLaM` which improves the function value estimation within each sub-region, and uses a latent representation of the search space. Empirically, `PlaLaM` outperforms existing path planning methods in 2D navigation tasks, especially in the presence of difficult-to-escape local optima, and shows benefits when plugged into the planning components of model-based RL such as PETS [7]. These gains transfer to highly multimodal real-world tasks, where we outperform strong baselines in compiler phase ordering by up to 245% and in molecular design by up to 0.4 on properties on a 0-1 scale.

1 Introduction

Path planning has been used extensively in many applications, ranging from reinforcement learning [7, 13, 14] and robotics [27, 35, 26] to biology [24], chemistry [39], material design [21], and compiler optimization [41]. The goal is to find the most rewarding trajectory (i.e., state-action sequence) $\mathbf{x} = (s_0, a_0, s_1, \ldots, s_n)$ in the search space Ω: $\mathbf{x}^* = \arg\max_{\mathbf{x}\in\Omega} f(\mathbf{x})$, where $f(\mathbf{x})$ is the reward.

In this work, we focus on path planning problems with long trajectories $\mathbf{x}$ and discontinuous and/or multimodal reward functions f. Such high-dimensional non-convex optimization problems exist in many real domains, both continuous and discrete. While we could always find near-optimal $\mathbf{x}$ by random sampling if given an infinite query budget, in practice it is preferred to have a sample-efficient method that achieves high-reward trajectories with the fewest queries of the reward function f.

While global methods like Bayesian Optimization (BO) [3] may struggle with limited samples and high-dimensional spaces, classic approaches like CEM [37] and CMA-ES [16] learn a local model around promising trajectories. For example, CEM tracks a population of trajectories and repeatedly resamples its population according to the highest-performing trajectories from the previous generation. On the other hand, such a focus can trap CEM in local optima, as confirmed empirically (Sec. 5).

Other recent approaches, such as VOOT [22] and DOO [31], use a (recursive) region partitioning scheme: they split the search space Ω into sub-regions $\Omega = \Omega_1 \cup \ldots \cup \Omega_k$, then invest more samples into promising sub-regions while continuing to explore other regions via an *upper confidence bound* (UCB). While such exploration-exploitation procedures adaptively focus on promising sub-regions and lead to sub-linear regret and optimality guarantees, their *region partition* procedure is manually

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

designed by humans and remains non-adaptive. For example, DOO partitions the space with uniform axis-aligned grids and VOOT with Voronoi cells, both independent of the reward f to be optimized.

Recently, Wang et al. proposed LaNAS [45] and LaMCTS [44], which *adaptively* partition the search regions based on sampled function values, and focus on good regions. They achieve strong empirical performance on Neural Architecture Search (NAS) and black-box optimization, outperforming many existing methods including evolutionary algorithms and BO. Notably, in recent NeurIPS'20 black-box optimization challenges, two teams that use variants of LaMCTS ranked 3rd [38] and 8th [23].

In this paper, we provide a simple theoretical analysis of LaMCTS to reveal the underlying principles of adaptive region partitioning, an analysis missing in the original work. Based on this analysis, we propose `PlaLaM`, a novel optimization technique for path-planning. Unlike LaMCTS, `PlaLaM` uses a latent representation of the search space. Additionally, we use the maximum (instead of the mean) as the node score to improve sample efficiency, verified empirically in Sec. 5.3. Both changes are motivated by our theoretical analysis.

We verify `PlaLaM` on several challenging path-planning tasks, including 2D navigation environments from past work with difficult-to-escape local optima, and real-world planning problems in compiler optimization and molecular design. In all tasks, `PlaLaM` demonstrates substantially stronger exploration ability to escape from local optima compared to multiple path-planning baselines including CEM, CMA-ES and VOOT. On compiler phase ordering, we achieve up to 245% speedup in execution cycles comparing to -O3 optimization and OpenTuner [1], two widely used optimization techniques in compilers. On molecular design, `PlaLaM` outperforms all of our baselines in generating molecules with high values of desirable properties, beating the best baseline in average property value by up to 0.4 on properties in a $[0, 1]$ range. Additionally, extensive ablation studies show factors that affect the quality of planning and verify the theoretical analysis.

`PlaLaM` is a general planning technique and can be readily plugged into existing algorithms with path planning components. For example, we apply `PlaLaM` to PETS [7] in model-based RL and observe substantially improved performance for high-dimensional continuous control and navigation, compared to CEM as used in the original PETS framework.

2 Latent Space Monte Carlo Tree Search (LaMCTS)

LaMCTS [44] is recently proposed to solve black-box optimization problems $\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x})$ via recursively learning f-dependent region partitions. Fig. 1 and Alg. 1 show the details of LaMCTS as well as our proposed approach `PlaLaM` (formally introduced in Sec. 4) for comparison.

LaMCTS starts with N_{init} random samples of the entire search space Ω (line 5 in Alg. 1). For a region Ω_k, let $n(\Omega_k)$ be the number of samples within. LaMCTS dictates that, if $n(\Omega_k) \geq N_{\text{thres}}$,

Algorithm 1 `PlaLaM` Pseudocode for Path Planning. Improvements over LaMCTS in green.

1: **Input:** Number of rounds T, Environment Oracle: $f(\mathbf{x})$, Dataset $\mathcal{D}$, Sampling Latent Model $h(\mathbf{x})$, Partitioning Latent Model $s(\mathbf{x})$.
2: **Parameters:** Initial #samples N_{init}, Re-partitioning interval N_{par}, Node partition threshold N_{thres}, UCB parameter C_p.
3: Pre-train $h(\cdot)$ on $\mathcal{D}$ when $\mathcal{D} \neq \emptyset$.
4: Set region partition $\mathcal{V}_0 = \{\Omega\}$.
5: Draw N_{init} samples uniformly from $\mathcal{S}_0 = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^{N_{\text{init}}} \subset \Omega$.
6: **for** $t = 0, \ldots, T - N_{\text{init}} - 1$ **do**
7: **if** t divides N_{par} **then**
8: Train/fine-tune latent model $h(\cdot)$ using samples $\mathcal{S}_t \cup \mathcal{D}$ (Eqn. 2).
9: Re-learn region partition $\mathcal{V}_t \leftarrow \text{Partition}(\Omega, \mathcal{S}_t, N_{\text{thres}}, s(\cdot))$ in latent space Φ_s of $s(\cdot)$.
10: **end if**
11: **for** $k := \text{root}, k \notin \mathcal{V}_{\text{leaf}}$ **do**
12: $k \leftarrow \arg\max_{\Omega_c \in \text{child}(\Omega_k)} b_c$, where $b_c := \left[\cancel{\frac{1}{n(\Omega_c)} \sum_{\mathbf{x}_i \in \Omega_c} f(\mathbf{x}_i)} \max_{\mathbf{x}_i \in \Omega_c} f(\mathbf{x}_i) + C_p \sqrt{\frac{2 \log n(\Omega_k)}{n(\Omega_c)}} \right]$.
13: **end for**
14: Initialize CMA-ES using encodings of $\mathcal{S}_t \cap \Omega_k$ via $h(\cdot)$. Here Ω_k is the chosen leaf sub-region.
15: $\mathcal{S}_t \leftarrow \mathcal{S}_{t-1} \cup \{(\mathbf{x}_t, f(\mathbf{x}_t))\}$, where $\mathbf{x}_t$ is drawn from CMA-ES and decoded via $h^{-1}(\cdot)$.
16: **end for**

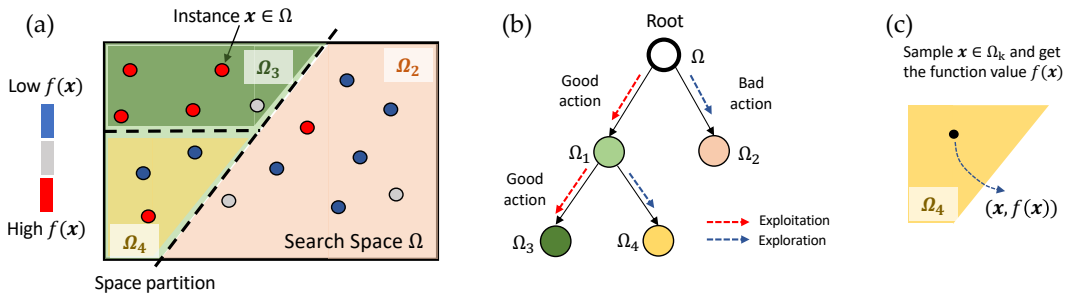


Figure 1: `PlaLaM` extends LaMCTS [44] to path planning. **(a)** Starting from a search space Ω, both `PlaLaM` and LaMCTS first draw a few samples $\mathbf{x} \in \Omega$, then learn to partition Ω into a sub-region Ω_1 with good samples (high $f(\mathbf{x})$) and a sub-region Ω_2 with bad samples (low $f(\mathbf{x})$). Compared to LaMCTS, `PlaLaM` uses a *latent space* and reduces the dimensionality of the search space. **(b)** Sampling follows the learned recursive space partition, focusing on good regions while still exploring bad regions using UCB. `PlaLaM` uses the *maximum* of the sampled value in a region ($\max_{\mathbf{x}_i \in \Omega} f(\mathbf{x}_i)$) as the node value, while LaMCTS uses the mean. **(c)** Upon reaching a leaf, new data points are sampled within the region and the space partition is relearned.

then Ω_k is partitioned into disjoint sub-regions $\Omega_k = \Omega_{\text{good}} \cup \Omega_{\text{bad}}$ as its *children* (Fig. 1(a)-(b), line 9 in Alg. 1, the function Partition). Intuitively, Ω_{good} contains promising samples with high f, while Ω_{bad} contains samples with low f. Unlike DOO and VOOT, such a partition is learned using $\mathcal{S}_t \cap \Omega_k$, our samples so far in the region, and is thus dependent on the function f to be optimized.

Given tree-structured sub-regions, new samples are mostly drawn from promising regions and occasionally from other regions for exploration. This is achieved by Monte Carlo Tree Search (MCTS) [4] (line 11-13): at each tree branching, the UCB score b is computed to balance exploration and exploitation (line 12). Then the subregion with highest UCB score is selected (e.g., it may have high f and/or low n). This is done recursively until a leaf sub-region Ω' is reached. Then a new sample $\mathbf{x}$ is drawn from Ω' (line 15) either uniformly, or from a local model constructed by an existing optimizer (e.g., TuRBO [10], CMA-ES [16]), in which case LaMCTS becomes a meta-algorithm. When more samples are collected, regions are further partitioned and the tree gets deeper.

Finally, the function Partition in Alg. 1 is defined as follows: first a 2-class K-means on $(\mathbf{x}, f(\mathbf{x}))$ is used to create positive/negative sample groups. Next, a SVM classifier is used to learn the decision boundary (hence the partition), so that samples with high $f(\mathbf{x})$ fall into Ω_{good}, and samples with low $f(\mathbf{x})$ fall into Ω_{bad} (Fig. 1(a)). See Appendix A for the pseudo code. The partition boundary can also be re-learned after more samples are collected (line 9).

3 A Theoretical Understanding of Space Partitioning

While LaMCTS [44] shows strong empirical performance, it contains several components with no clear theoretical justification. Here we attempt to give a formal regret analysis when sub-regions $\{\Omega_k\}$ are *fixed* and all at the same tree level, and the function f is deterministic. We leave further analysis of tree node splitting and evolution of hierarchical structure to future work.

Despite the drastic simplification, our regret bound still shows why an f-dependent region partition is helpful. By showing that a better regret bound can be achieved by a clever region partition as empirically used in the Partition function in Alg. 1, we justify the design of LaMCTS. Furthermore, our analysis suggests several empirical improvements over LaMCTS and motivates the design of `PlaLaM`, which outperforms multiple classic approaches on hard path planning problems.

3.1 Regret Analysis with Fixed Sub-Regions

We consider the following setting. Suppose we have K d-dimensional regions $\{\Omega_k\}_{k=1}^K$, and $n_t(\Omega_k)$ is the visitation count at iteration t. The global optimum $\mathbf{x}^*$ resides in some unknown region Ω_{k^*}. At each iteration t, we visit a region Ω_k, sample (uniformly or otherwise) a data point $\mathbf{x}_t \in \Omega_k$, and retrieve its *deterministic* function value $f_t = f(\mathbf{x}_t)$. In each region Ω_k, define $\mathbf{x}_k^* := \arg\max_{\mathbf{x} \in \Omega_k} f(\mathbf{x})$ and the maximal value $g^*(\Omega_k) = f(\mathbf{x}_k^*)$. The maximal value *so far* at iteration t is $g_t(\Omega_k) = \max_{t' \leq t} f(\mathbf{x}_{t'})$. It is clear that $g_t \leq g^*$ and $g_t \to g^*$ when $t \to +\infty$.

We define the *confidence bound* $r_t = r_t(\Omega_k)$ so that with high probability, the following holds:

$$g_t(\Omega_k) \geq g^*(\Omega_k) - r_t(\Omega_k) \tag{1}$$

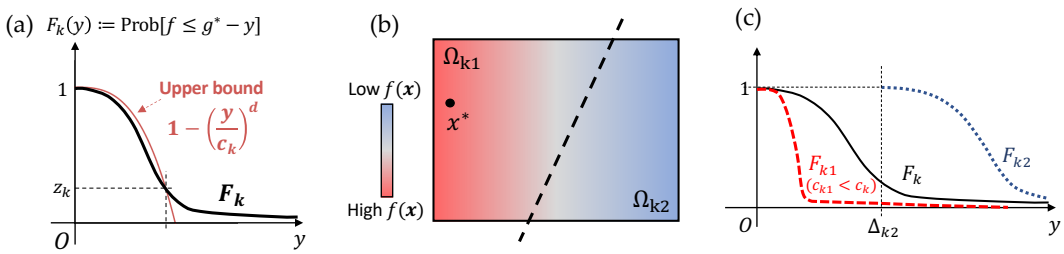


Figure 2: Theoretical understanding of space partitioning. **(a)** Definition of (z_k, c_k)-diluted region Ω_k (Def. 1). **(b)** Partition of region Ω_k into good region Ω_{k1} and bad region Ω_{k2}. Optimal solution $\mathbf{x}^* \in \Omega_{k1}$. **(c)** After space partitioning, F_k is split into F_{k1} and F_{k2}. The good region F_{k1} has much smaller c_{k1} while the bad region has much larger best-to-optimality gap Δ_{k2}. As a result, the expected total regret decreases.

At iteration t, we pick region k_t to sample based on the upper confidence bound: $k_t = \arg\max_k g_t(\Omega_k) + r_t(\Omega_k)$. Many different confidence bounds can be applied; for convenience in this analysis, we use the "ground truth" bound from the cumulative density function (CDF) of f within the region Ω_k (Please check Appendix B for all proofs):

Lemma 1. *Let $F_k(y) \coloneqq \mathbb{P}\left[f(\mathbf{x}) \le g^*(\Omega_k) - y | \mathbf{x} \in \Omega_k\right]$ be a strictly decreasing function, and let $r_{k,t}(\Omega_k) \coloneqq F_k^{-1}\left(\delta^{1/n_t(\Omega_k)}\right)$. Then Eqn. 1 holds with probability $1-\delta$.*

Here F_k^{-1} is the inverse function of F_k and randomness arises from sampling within Ω_k. Since F_k is a strictly decreasing function, F_k^{-1} exists and is also strictly decreasing. By definition, $F_k \in [0,1]$, $F_k(0) = 1$ and $F_k^{-1}(1) = 0$. We then define the *dilution* of each region as follows:

Definition 1 ((z_k, c_k)-dilution)**.** *A region Ω_k is (z_k, c_k)-diluted if there exist z_k, c_k such that $F_k(y) \le 1 - (y/c_k)^d$ for $y \in [0, c_k(1-z_k)^{1/d}]$, where z_k is the smallest $F_k(y)$ to make the inequality hold.*

See Fig. 2(a) for visualization. Obviously if Ω_k is (z_k, c_k)-diluted then it is (z'_k, c'_k)-diluted for any $c'_k \ge c_k$ and $z'_k \ge z_k$. Therefore, we often look for the smallest z_k and c_k to satisfy the condition. If a region Ω_k has small c_k and z_k, we say it is *highly concentrated*. For example, if $f(\mathbf{x})$ is mostly constant within a region, then c_k is very small since $F_k(y)$ drops to 0 very quickly.

While the definition of concentration may be abstract, we show it is implied by Lipschitz continuity:

Corollary 1. *If a region Ω_k is L_k-Lipschitz continuous, i.e., $|f(\mathbf{x}) - f(\mathbf{x}')| \le L_k \|\mathbf{x} - \mathbf{x}'\|_2$, and there exists an ϵ_0-ball $B(\mathbf{x}_k^*, \epsilon_0) \subseteq \Omega_k$, then with uniform sampling, Ω_k is $(1 - \epsilon_0^d \tilde{V}_k^{-1}, L_k \sqrt[d]{\tilde{V}_k})$-diluted. Here $\tilde{V}_k \coloneqq V_k / V_0$ is the relative volume with respect to the unit sphere volume V_0.*

Typically, a smoother function (with small L_k) and large ϵ_0 yield a less diluted (and more concentrated) region. However, the concept of dilution (Def. 1) is much broader. For example, if we shuffle function values within Ω_k, Lipschitz continuity is likely to break but Def. 1 still holds.

Now we will bound the total regret. Let $R_t(a_t) \coloneqq f^* - g_t(\Omega_{a_t}) \ge 0$ be the regret of picking Ω_{a_t} and $R(T) \coloneqq \sum_{t=1}^T R_t(a_t)$ be the total regret, where T is the total number of samples (queries to f). Define the *gap* of each region $\Delta_k \coloneqq f^* - g^*(\Omega_k)$ and split the region indices into $\mathcal{K}_{\text{good}} \coloneqq \{k : \Delta_k \le \Delta_0\}$ and $\mathcal{K}_{\text{bad}} \coloneqq \{k : \Delta_k \ge \Delta_0\}$ by a threshold Δ_0. $C_{\text{good}} \coloneqq \left(\sum_{k \in \mathcal{K}_{\text{good}}} c_k^d\right)^{1/d}$ and $C_{\text{bad}} \coloneqq \left(\sum_{k \in \mathcal{K}_{\text{bad}}} c_k^d\right)^{1/d}$ are the ℓ_d-norms of the c_k in these two sets. Finally, $M \coloneqq \sup_{\mathbf{x} \in \Omega} f(\mathbf{x}) - \inf_{\mathbf{x} \in \Omega} f(\mathbf{x})$ is the maximal gap between function values. Treating each region Ω_k as an arm and applying a regret analysis similar to multi-arm bandits [40], we obtain the following theorem:

Theorem 1. *Suppose all $\{\Omega_k\}$ are (z_k, c_k)-diluted with $z_k \le \eta / T^3$ for some $\eta > 0$. The total expected regret $\mathbb{E}[R(T)] = \mathcal{O}\left[C_{\text{good}} \sqrt[d]{T^{d-1} \ln T} + M (C_{\text{bad}}/\Delta_0)^d \ln T + KM\eta/T\right]$.*

3.2 Implications of Theorem 1

The effect of space partitioning. Reducing $\{c_k\}$ results in a smaller regret $R(T)$. Thus if we can partition Ω_k into two sub-regions Ω_{k1} and Ω_{k2} such that the good partition Ω_{k1} has smaller $c_{k1} < c_k$ and the bad partition Ω_{k2} has larger $\Delta_{k2} > \Delta_0$ and falls into $\mathcal{K}_{\text{bad}}$, then we can improve the regret

bound (Fig. 2(b)-(c)). This coincides with the $\mathrm{Partition}$ function of LaMCTS very well: it samples a few points in Ω_k, and trains a classifier to separate high f from low f. On the other hand, if we partition a region Ω_k randomly, e.g., each $f(\mathbf{x})$ is assigned to either Ω_{k1} or Ω_{k2} at random, then statistically $F_{k1} = F_{k2} = F_k$ and $c_{k1} = c_{k2} = c_k$, which *increases* the regret bound. Therefore, the partition needs to be *informed* by data that have already been sampled within the region Ω_k.

Recursive region partitioning. In Theorem 1, we assume all regions $\{\Omega_k\}$ have fixed c_k and z_k, so the bound breaks for large enough T (as η/T^3 eventually becomes smaller than any fixed z_k). However, as LaMCTS conducts further internal partitioning within Ω_k, its c_k and z_k keep shrinking with more samples T. If each split leads to slightly fewer bad f (i.e., lighter "tail"), with the ratio being $\gamma < 1$, then by the definition of CDF, z_k is the probability mass of the tail and thus $z_k \sim \gamma^{-T/N_{\mathrm{par}}}$. This would yield $z_k \leq \eta/T^3$ for all T, since γ^{-T} decays faster than $1/T^3$ and Theorem 1 would hold for all T. See Appendix E.2 for empirical verification of decaying z_k.

3.3 Related Work and Limitations

While related to Lipschitz bandits [28] and coarse-to-fine deterministic function optimization like DOO and SOO [32], our analysis is fundamentally different. We have discussed how f-dependent region partitioning and a data-driven learning procedure affect the regret bound, which to our knowledge has not been addressed in previous works. See Appendix B.5 for additional remarks on Theorem 1.

There is more work to be done to fully understand how LaMCTS works. In particular, we did not analyze when to split a node (e.g. how many samples we need to collect before making a decision), or the effect of relearning the space partition. We leave these to future work.

4 `PlaLaM` for Path Planning

Based on our analysis, we propose `PlaLaM`, which extends LaMCTS to path planning, a problem with temporal structure. `PlaLaM` outperforms baseline path planning approaches in both continuous and discrete path planning problems. Here we represent trajectories as action sequences $\mathbf{x} = (a_0, a_1, \ldots, a_{n-1})$ and treat them as high-dimensional vectors $\mathbf{x}$ in the trajectory space Ω. We then search for the best trajectory in Ω.

Thus, `PlaLaM` searches over the space Ω, recursively partitioning Ω into subregions based on trajectory reward, and sampling from subregions using CMA-ES [16] (which is faster than TuRBO [10] used in the original LaMCTS). We emphasize again that `PlaLaM`'s region partitioning procedure is fully adaptive, in contrast to traditional MCTS approaches such as VOOT, which only partition the trajectory space based on one action at a time.

Additionally, we have made several improvements over the original LaMCTS, as detailed in Algorithm 1. First, we use the maximal value $\max_{i \in \Omega_k} f(\mathbf{x}_i)$ rather than the mean value $\frac{1}{n(\Omega_k)} \sum_{i \in \Omega_k} f(\mathbf{x}_i)$ as the metric of goodness for each node k (and its associated region Ω_k). This is driven by Theorem 1, which gives a regret bound based on maximum values. Intuitively, using the mean value would cause the algorithm to be slow to respond to newly discovered territory: it takes time for the mean metric to boost, and we may miss important leaves. We show the difference empirically in Sec. 5.3.

Second, Theorem 1 suggests that a lower-dimensional (smaller d) and smoother (smaller c_k) representation leads to lower regret. Therefore, `PlaLaM` employs a latent space as described below.

4.1 Latent Spaces For Sampling and Partitioning

We can find a latent representation for $\mathbf{x}$ by mapping the space Ω to and from a latent space Φ_h, using an encoder h and decoder h^{-1}. h could be a principal component analysis (PCA) [48], a random network encoding [42], or a reversible flow [9], depending on the environment's particular Ω and state/action structure. While some latent representations can be fixed by specifying the inductive bias (e.g., random network encoding), others can be learned from data, optimizing reconstruction loss:

$$\min_h \mathbb{E}_{\mathbf{x}} \left[w(\mathbf{x}) \| h^{-1}(h(\mathbf{x})) - \mathbf{x} \|^2 \right] \tag{2}$$

where $w(\mathbf{x})$ is a weighting function emphasizing trajectories with high cumulative reward $f(\mathbf{x})$. In this case, h and h^{-1} may be fine-tuned using each new $(\mathbf{x}, f(\mathbf{x}))$ pair when PlaLaM proposes and queries a new trajectory $\mathbf{x}$, or they may be pre-trained using a set of unlabeled $\mathbf{x}$ with $w(\mathbf{x}) \equiv 1$.

Additionally, PlaLaM can leverage a second latent space Φ_s with a second encoder s for the *partition* space. That is, we disentangle the *sampling* space Φ_h from which we sample new candidate trajectories, from the *partition* space Φ_s on which we construct the search space partition. Critically, we do not need s^{-1}: we never decode from Φ_s back to Ω. Thus s can dramatically reduce the dimension of the partition space, which may improve regularization due to the small number of samples, without suffering large reconstruction loss. Although s will be fixed rather than learned in this case, one may e.g., choose $s(\mathbf{x})$ to be a periodic snapshot of the state. Once the partition has been constructed on Φ_s, and we select a leaf region to propose from, we sample new $\mathbf{x}$ from Φ_h as before.

5 PlaLaM on Synthetic Environments

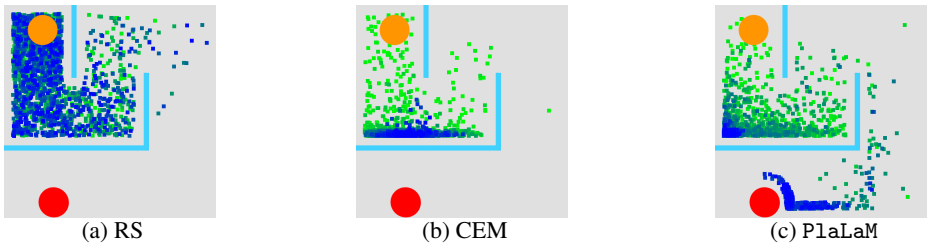
We test PlaLaM on a diverse set of environments to evaluate its performance in different settings.

Baselines. We compare PlaLaM to several baselines. **Random Shooting (RS)** [36] samples random trajectories and returns the best one. **Cross-Entropy Methods (CEM)** [2] use the top-k samples to fit a local model to guide future sampling. A related approach, **Covariance matrix adaptation evolution strategy (CMA-ES)** [16], tracks additional variables for improved local model fitting. **Voronoi optimistic optimization applied to trees (VOOT)** [22] is a "traditional" MCTS method for continuous action spaces that builds a tree on actions at each timestep. Finally, **iLQR** [26] is a seminal gradient-based local optimization approach used extensively in controls.

PlaLaM does not require substantially more tuning effort than CEM or CMA-ES, the best-performing among our baselines experimentally. The only additional hyperparameter tuned in PlaLaM is the C_p controlling exploration when selecting regions to sample from, which is dependent on the scale of the reward function. However, our C_p only varies by a factor of up to 10 across our diverse environments, and performance is not overly sensitive to small changes (Appendix E.4).

We use MiniWorld [5] for continuous path planning and MiniGrid [6] for discrete.

5.1 MiniWorld

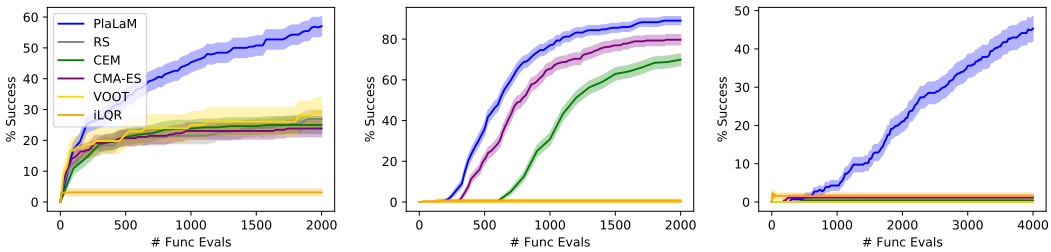


(a) RS (b) CEM (c) PlaLaM

Figure 3: MazeS3 environment. **Start:** Orange circle. **Goal:** Red square. Dots indicate final agent positions of 2,000 proposed trajectories (green: first iteration, blue: last iteration). CEM gets stuck in a local optimum of reward (shown as concentration of blue dots), while PlaLaM succeeds in reaching the goal.

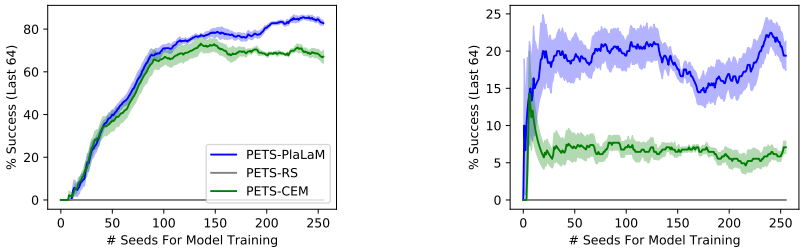
We consider the following 2D navigation tasks in *MiniWorld*. **MazeS3**: Agent navigates in a 3 by 3 maze to a goal. Greedy path planning gets stuck in local optima (Figure 3). **FourRooms**: Agent navigates from one room in a 2 by 2 configuration to a goal in the diagonally opposite room. Greedy path planning gets stuck in a corner. **SelectObj**: Open space with two goals. Large final reward when reaching the farther goal, while a distance-based reward misleadingly points to the closer goal. For full environment specifics, see Appendix G.1.

We modify the original setup to use a continuous action space (Δx and Δy), and provide a sparse reward (proximity to goal, with an additional bonus for reaching the goal) at end-of-episode. We use a high-dimensional top-down image view as the state. We featurize this image using a randomly initialized convolutional neural network, a reasonable feature extractor as shown in [42]. PlaLaM uses periodic snapshots of the featurized state as the partition space Φ_s, while Φ_h is the identity. Success is defined using a binary indicator for reaching the goal (far goal for SelectObj).



(a) MazeS3 (b) FourRooms (c) SelectObj

Figure 4: Mean, and standard deviation of mean (256 trials; fewer for VOOT due to speed), of success rate across MiniWorld tasks. `PlaLaM` significantly outperforms all baselines on all three tasks.



(a) PETS FourRooms (b) PETS SelectObj

Figure 5: `PlaLaM` in PETS compared to original PETS planners on MiniWorld environments, using PETS-learned world models. Sliding length-64 window of success percentage against number of training seeds for world model. `PlaLaM` significantly outperforms all baselines on both tasks.

Results. `PlaLaM` substantially outperforms all baselines on all three tasks, despite heavily tuning the baselines hyperparameters (Appendix F), showing that `PlaLaM` works for challenging tasks containing suboptimal local maxima. In MazeS3, `PlaLaM` succeeds but CEM gets stuck (Figure 3). VOOT, which builds an MCTS tree on actions at each timestep, struggles on all environments; `PlaLaM` can be viewed as an extension of MCTS that performs better on such long-horizon tasks. In the most difficult SelectObj task, `PlaLaM` solves nearly half of environment seeds within 4,000 queries of the oracle, whereas baselines quickly reach the near goal but struggle to escape this local optimum.

We also evaluate `PlaLaM` when combined with a model-based approach, PETS [7], on FourRooms and SelectObj (omitting MazeS3 because the changing maze walls for each seed make it difficult to learn a world model). Following PETS' setting and due to difficulty in learning image-based world models [12, 14], we use 2D agent position as the state. As shown in Fig. 5, `PlaLaM` substantially outperforms the authors' original CEM implementation in the PETS framework, demonstrating that it is not reliant on access to the oracle model but can work with learned models as well.

5.2 MiniGrid

MiniGrid [6] is a popular sparse-reward symbolic environment for benchmarking RL algorithms. It contains tasks with discrete states and actions such as **DoorKey (DK)**: pick up a key and open the door connecting two rooms; **MultiRoom (MR)**: traverse several rooms by opening doors; and **KeyCorridor (KC)**, a combination of MR and DK: some doors are locked and require a key. As in MiniWorld, we add proximity to the goal to the final sparse reward.

In discrete action spaces, `PlaLaM` optimizes the vector of all action probabilities over all timesteps, and takes the highest-probability action at each step. As in MiniWorld, we use periodic state snapshots featurized by a randomly initialized CNN as the partition space Φ_s. The sampling space Φ_h is given by a reversible flow model [9] trained on `PlaLaM`'s generated trajectories. We compare `PlaLaM` to the same baselines as in MiniWorld, except VOOT and iLQR which are designed for continuous tasks.

Results. `PlaLaM` is equal to or better than baselines on all six tasks (Table 1). Especially in the hardest tasks with the most rooms (MR-N4S5, MR-N6), `PlaLaM` improves substantially over baselines.

	DK-6	DK-8	KC-S3R3	KC-S3R4	MR-N4S5	MR-N6
RS	**0.97±0.01**	0.34±0.13	-2.38±0.09	**-4.27±0.12**	-18.16±0.80	-119.39±4.64
CEM	0.03±0.12	-3.34±0.34	-3.40±0.08	-4.93±0.13	-22.88±1.00	-131.32±5.24
CMAES	0.93±0.03	0.23±0.14	-2.46±0.09	-4.44±0.12	-14.31±0.78	-117.50±4.61
PlaLaM	**0.97±0.02**	**0.48±0.11**	**-2.19±0.15**	**-4.22±0.13**	**-10.68±0.68**	**-112.72±4.46**

Table 1: Results for PlaLaM in MiniGrid. PlaLaM is equal or better on all tasks.

5.3 Analysis

We run ablations on PlaLaM in MiniWorld to justify our methodological choices. See Appendix E for more complete analysis.

Region Selection in PlaLaM. We consider three different region selection methods. (1) PlaLaM-mean: using mean function value rather than max, as in LaMCTS [44]; **(2)** PlaLaM-notree: directly selecting the leaf with the highest UCB score; and **(3)** PlaLaM-noUCB: only using node value rather than UCB. PlaLaM greatly outperforms all three variations in MiniWorld, which justifies our design.

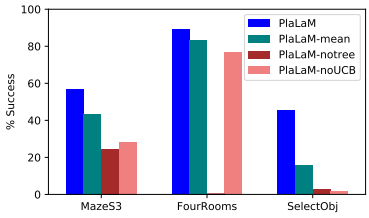


Figure 6: MiniWorld success percentages with different region selection methods.

	MazeS3	FourRooms	SelectObj
L_k	87.5	100.0	100.0
c_k	81.3	93.8	100.0

Table 2: Percentage out of 32 environment seeds on MiniWorld environments where PlaLaM yields a better estimated Lipschitz and c_k compared to random partitioning on the same nodes.

Data-driven space partition in PlaLaM **vs. random partitioning**. We examine c_k in Def. 1 and Lipschitz constant L_k in Corollary 1 to verify the theory. We conduct a preliminary analysis on PlaLaM's tree after the full 2,000 queries (4,000 for SelectObj). At each intermediate node, we estimate L_k and c_k of its children from the PlaLaM partition, against a random partition that divides the node's samples with the same ratio (see Appendix E.1 for estimation details). We then average the values for both PlaLaM and random partitions over all nodes in the tree. We find that PlaLaM does yield lower average L_k and c_k (Table 2), indicating that our data-driven space partition is effective.

6 PlaLaM on Real-World Applications

6.1 Compiler Phase Ordering

Compiler optimization applies a series of program transformations from a set of predefined optimizations (e.g., *loop invariant code motion (LICM)*, *function inlining* [30]) to improve code performance. Since these optimizations are not commutative, the order in which they are applied is extremely important. This problem, known as *phase ordering*, is a core challenge in the compiler community.

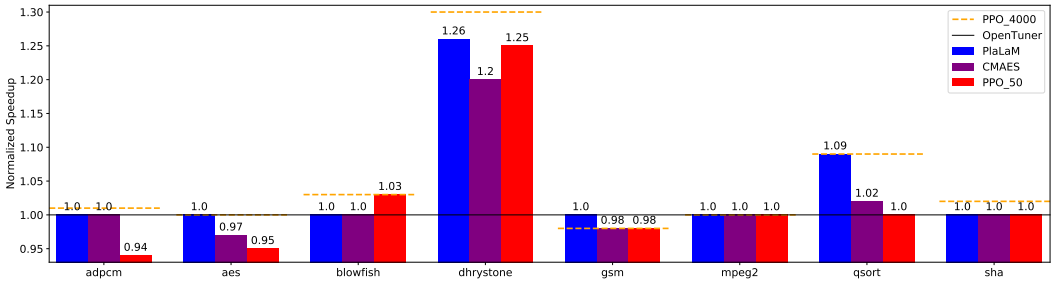
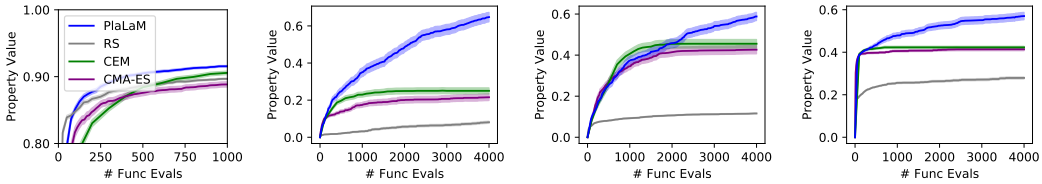


Figure 7: Compiler phase ordering results, in terms of normalized execution cycles with respect to OpenTuner [1], a widely used method for program autotuning. PlaLaM is consistently equal or better compared to baselines. We omitted the matmul task since it doesn't fit the scale with its 245% speedup over OpenTuner.



(a) QED (b) DRD2 (c) HIV (d) SARS

Figure 8: Mean and standard deviation (128 trials), of max property value discovered in molecular design tasks. `PlaLaM` significantly outperforms all baselines on all properties.

Current solutions to this NP-hard problem rely heavily on heuristics: groups of optimizations are often packed into "optimization levels" (such as -O3 or -O0) hand-picked by developers [34, 41].

We apply `PlaLaM` to the standard CHStone benchmarks [17], and use periodic snapshots of states as Φ_s and the identity as Φ_h. See Appendix G.2 for full environment details.

Results. Compared to -O3 and -O0 (the standard compiler optimization levels), `PlaLaM` achieves 61% to 245% speedup. Compared to a stronger PPO baseline using 50 samples (PPO_50) and to CMA-ES, we achieve up to 10% and 7% speedup respectively. Finally, compared to final PPO results at convergence after 4000 samples (PPO_4000) as an "optimal" number, `PlaLaM` does similarly on most tasks. We refer the reader to Appendix D for the detailed results.

6.2 Molecular Design

Finally, we evaluate `PlaLaM` on molecular design. Given an oracle for a desired molecular property, the goal is to generate molecules with high property score after the fewest trials. This is critical to pharmaceutical drug development [43], as property evaluations require expensive wet-lab assays.

Similar to [18], we fix a query budget and optimize several properties: **QED**: a synthetic measure of drug-likeness, relatively simpler to optimize; **DRD2**: a measure of binding affinity to a human dopamine receptor; **HIV**, the probability of inhibition potential for HIV; and **SARS**: the same probability for a variant of the SARS virus, related to the SARS-CoV-2 virus responsible for COVID-19. All four properties have a range of $[0, 1]$; higher is better. For DRD2, HIV, and SARS, we evaluate using computational predictors from [33] (DRD2) and [50] (HIV, SARS) in lieu of wet-lab assays.

To run `PlaLaM` on molecular design, we view the molecular string representation (SMILES string [47]) as the action sequence, similar to how many generative models generate molecules autoregressively [11, 25, 8, 49]. Following the state-of-the-art HierG2G model from [19], we learn a latent representation from a subset of ChEMBL [29], a dataset of 1.8 million drug-like molecules, *without* using any of its property labels (e.g., effectiveness in binding to a particular receptor). During this unsupervised training, we only use the 500k molecules with the lowest property scores to ensure a good molecule is discovered by search rather than a simple retrieval from the dataset. Our setting differs from many existing methods for molecular design, which assume a large preexisting set of molecules with the desired property for training the generator [33, 20, 51, 49].

The latent space is used as both the sampling space Φ_h and partition space Φ_s for `PlaLaM`. Since it is trained on additional unlabeled data, all baselines operate in the same space for fair comparison.

Results. Figure 8 shows the highest property score discovered by each method for each property. The absolute difference is small in the relatively simple synthetic QED task. However, `PlaLaM` outperforms all baselines by a much greater margin—up to 0.4 in DRD2—in the more challenging and realistic DRD2, HIV, and SARS tasks, where CEM and CMA-ES quickly plateau but `PlaLaM` continues to improve with more function evaluations.

7 Conclusion

We propose `PlaLaM`, a novel algorithm for path planning that learns to partition the search space so that subsequent sampling focuses more on promising regions. We provide a formal regret analysis of region partitioning, motivating improvements that yield large empirical gains. `PlaLaM` particularly excels in environments with difficult-to-escape local optima, substantially outperforming strong baselines on 2D navigation tasks as well as real-world compiler optimization and molecular design.

References

[1] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316, 2014.

[2] Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

[3] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[4] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[5] Maxime Chevalier-Boisvert. gym-miniworld environment for openai gym. `https://github.com/maximecb/gym-miniworld`, 2018.

[6] Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *NeurIPS*, 2018.

[8] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *arXiv preprint arXiv:1802.08786*, 2018.

[9] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[10] David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. *arXiv preprint arXiv:1910.01739*, 2019.

[11] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[12] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[13] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[14] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[15] Ameer Haj-Ali, Qijing Jenny Huang, John Xiang, William Moses, Krste Asanovic, John Wawrzynek, and Ion Stoica. Autophase: Juggling hls phase orderings in random forests with deep reinforcement learning. *Proceedings of Machine Learning and Systems*, 2:70–81, 2020.

[16] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[17] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, Hiroaki Takada, and Katsuya Ishii. Chstone: A benchmark program suite for practical c-based high-level synthesis. In *2008 IEEE International Symposium on Circuits and Systems*, pages 1192–1195. IEEE, 2008.

[18] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2323–2332. PMLR, 2018.

[19] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, pages 4839–4848. PMLR, 2020.

[20] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv preprint arXiv:1812.01070*, 2018.

[21] Seiji Kajita, Tomoyuki Kinjo, and Tomoki Nishi. Autonomous molecular design by monte-carlo tree search and rapid evaluations using molecular dynamics simulations. *Communications Physics*, 3(1):1–11, 2020.

[22] Beomjoon Kim, Kyungjae Lee, Sungbin Lim, Leslie Kaelbling, and Tomas 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.

[23] Taehyeon Kim, Jaeyeon Ahn, Nakyil Kim, and Seyoung Yun. Adaptive local bayesian optimization over multiple discrete variables. *Workshop at NeurIPS 2020 Competition Track on Black-Box Optimization Challenge*, 2020.

[24] Christian Kroer and Tuomas Sandholm. Sequential planning for steering immune system adaptation. In *IJCAI*, pages 3177–3184, 2016.

[25] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–1954. PMLR, 2017.

[26] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229. Citeseer, 2004.

[27] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016.

[28] Stefan Magureanu, Richard Combes, and Alexandre Proutiere. Lipschitz bandits: Regret lower bound and optimal algorithms. In Maria Florina Balcan, Vitaly Feldman, and Csaba Szepesvári, editors, *Proceedings of The 27th Conference on Learning Theory*, volume 35 of *Proceedings of Machine Learning Research*, pages 975–999, Barcelona, Spain, 13–15 Jun 2014. PMLR.

[29] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, María Gordillo-Marañón, Fiona Hunter, Laura Junco, Grace Mugumbate, Milagros Rodriguez-Lopez, Francis Atkinson, Nicolas Bosc, Chris J Radoux, Aldo Segura-Cabrera, Anne Hersey, and Andrew R Leach. ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Research*, 47(D1):D930–D940, 11 2018.

[30] Steven Muchnick et al. *Advanced compiler design implementation*. Morgan kaufmann, 1997.

[31] Rémi Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 783–791, Red Hook, NY, USA, 2011. Curran Associates Inc.

[32] Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. 2014.

[33] Marcus Olivecrona, Thomas Blaschke, Ola Engkvist, and Hongming Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):1–14, 2017.

[34] Zhelong Pan and Rudolf Eigenmann. Fast and effective orchestration of compiler optimizations for automatic performance tuning. In *International Symposium on Code Generation and Optimization (CGO'06)*, pages 12–pp. IEEE, 2006.

[35] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pages 489–494. IEEE, 2009.

[36] Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

[37] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. pages 127, 190, 1999.

[38] Mikita Sazanovich, Anastasiya Nikolskaya, Yury Belousov, and Aleksei Shpilman. Solving black-box optimization challenge via learning search space partition for local bayesian optimization. *workshop at NeurIPS 2020 Competition Track on Black-Box Optimization Challenge*, 2020.

[39] Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018.

[40] Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

[41] Spyridon Triantafyllis, Manish Vachharajani, Neil Vachharajani, and David I August. Compiler optimization-space exploration. In *International Symposium on Code Generation and Optimization, 2003. CGO 2003.*, pages 204–215. IEEE, 2003.

[42] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018.

[43] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature Reviews Drug Discovery*, 18(6):463–477, 2019.

[44] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. *NeurIPS*, 2020.

[45] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[46] Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.

[47] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

[48] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[49] Kevin Yang, Wengong Jin, Kyle Swanson, Regina Barzilay, and Tommi Jaakkola. Improving molecular design by stochastic iterative target augmentation. In *International Conference on Machine Learning*, pages 10716–10726. PMLR, 2020.

[50] Kevin Yang, Kyle Swanson, Wengong Jin, Connor Coley, Philipp Eiden, Hua Gao, Angel Guzman-Perez, Timothy Hopper, Brian Kelley, Miriam Mathea, et al. Analyzing learned molecular representations for property prediction. *Journal of chemical information and modeling*, 59(8):3370–3388, 2019.

[51] Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We claim to provide a theoretical explanation of region partitioning and empirical gains over baselines, which are presented in Sec. 3 and Secs. 5,6 respectively.
   (b) Did you describe the limitations of your work? [Yes] We have discussed limitations of our preliminary theory in Sec. 3.2. Our latent spaces also inherently depend on the details of the environments, as described in each individual experiment section. While `PlaLaM` could be easily modified for black-box optimization in principle, we have made clear that we empirically verify only on path planning.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not foresee any obvious negative societal impacts from our work, which contributes a general-purpose path planning algorithm.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] In lemma/theorem statements in Sec. 3.
   (b) Did you include complete proofs of all theoretical results? [Yes] All proofs are in Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We upload code in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We discuss all hyperparameter tuning details in Appendix F.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Included in all experiments in Secs. 5,6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] Just code, which is in supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use publicly available datasets/tasks.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We don't use data of this sort.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]