Curriculum Generation for Learning Guiding Functions in State-Space Search Algorithms

Sumedh Pendurkar[1], **Levi H.S. Lelis**[2, 3], **Nathan R. Sturtevant**[2, 3], **Guni Sharon**[1]

[1]Department of Computer Science and Engineering, Texas A&M University
[2]Department of Computing Science, University of Alberta
[3] Alberta Machine Intelligence Institute (Amii)
sumedhpendurkar@tamu.edu, levi.lelis@ualberta.ca, nathanst@ualberta.ca, guni@tamu.edu

Abstract

This paper investigates methods for training parameterized functions for guiding state-space search algorithms. Existing work commonly generates data for training such guiding functions by solving problem instances while leveraging the current version of the guiding function. As a result, as training progresses, the guided search algorithm can solve more difficult instances that are, in turn, used to further train the guiding function. These methods assume that a set of problem instances of varied difficulty is provided. Since previous work was not designed to distinguish the instances that the search algorithm can solve from those that cannot be solved with the current guiding function, the training method commonly wastes time attempting and failing to solve many of these instances. In this paper, we improve upon these training methods by generating a curriculum for learning the guiding function that directly addresses this issue. Namely, we propose and evaluate a Teacher-Student Curriculum (TSC) approach where the teacher is an evolutionary strategy that attempts to generate problem instances of "correct difficulty" and the student is a guided search algorithm utilizing the current guiding function. The student attempts to solve the problem instances generated by the teacher. We conclude with experiments demonstrating that TSC outperforms the current state-of-the-art Bootstrap Learning method in three representative benchmark domains with respect to the time required to solve all instances of the test set.

1 Introduction

This paper considers deterministic single-agent state-space search problems (referred to as *search problems*). Specifically, we focus on state-space search algorithms (referred to as *search algorithms*) which are guided by a function, denoted *guiding function*. This function is designed to guide a search algorithm to efficiently find a start-to-goal path in a given search space (e.g., a heuristic function). Existing work (Agostinelli et al. 2019; Orseau and Lelis 2021) uses machine learning based methods to train a parameterized guiding function and couple them with search algorithms. We refer to such algorithms as *guided search algorithms*.

Guided search algorithms have demonstrated state-of-the-art performance in several domains (Agostinelli et al. 2019; Orseau and Lelis 2021). Such algorithms can be roughly divided into two categories, namely, heuristic-based (Arfaee, Zilles, and Holte 2010; McAleer et al. 2018; Agostinelli et al. 2019, 2021) and policy-based (Orseau et al. 2018). Another way to characterize existing work is based on how the training/learning is performed. Learning is performed *offline* if the trained function is not used to obtain new training data (McAleer et al. 2018; Agostinelli et al. 2019, 2021). Similarly, learning is performed *online* if the training is a progressive procedure that interleaves between solving new instances and using the results to further train the function (Arfaee, Zilles, and Holte 2010; Orseau and Lelis 2021).

Previous online guided search algorithms commonly rely on a learning method referred to as *Bootstrap Learning* introduced by Arfaee, Zilles, and Holte (2010). This learning method requires a set of problem instances provided by the user. The set of problem instances should be varied in difficulty, that is, it should contain problem instances that range from "easy" to solve by the search algorithm to "difficult" to solve. However, attempting to solve "difficult" instances with a guided search algorithm when the guiding function is not well trained might not be feasible. Consequently, attempting to solve difficult instances using a poorly trained guiding function often fails, and thus does not produce training data to further train the guiding function. Additionally, it might also significantly slow the training process. As such, providing an ordered set of problem instances of "correct difficulty" can potentially help speed up the training process.

The aim of this paper is to improve the Bootstrap Learning method by introducing curriculum generation methods that address the gap discussed above. Specifically, we propose a learning method referred to as *Teacher-Student Curriculum* (TSC) inspired by similar methods in the reinforcement learning literature (Matiisen et al. 2019; Narvekar et al. 2020). The objective of the teacher is to generate "easier" instances at early training stages, and gradually generate "more difficult" instances following improved performance of the guided search algorithm. We use the number of random steps (i.e., length of random walk) from the goal as a proxy of the resulting search problem instance difficulty. Our experimental results show that TSC achieves 7.5 to 31 times faster training compared to the state-of-the-art Bootstrap Learning method when coupled with three guided search algorithms in three representative benchmark

domains. We also provide a scaling study that demonstrates a clear advantage for using our TSC approach as the complexity of search problems scales upwards. [1]

2 Preliminaries

First, we formally describe the state-space search problem along with the definitions related to the search algorithms. Next, we define the formal objective of the learning methods. Finally, we describe the existing Bootstrap Learning method and an enhanced version of a random walk based learning method.

2.1 State-Space Search Terminology

A state-space search problem Q is given by $\langle S, A, \mathcal{T} \rangle$ and a problem instance P is a (s_0, g) pair that is coupled with a search problem Q where

- S is the discrete state space
- A is the discrete action space
- $\mathcal{T} : S \times A \rightarrow S$ is a deterministic transition function. $\mathcal{T}(s, a)$ is the state that is a result of taking action a from state s.
- s_0 is the initial state, $s_0 \in S$
- g is the goal state, $g \in S$

A *solution* is a sequence of state-action pairs $\{(s_i, a_i)\}_{i \in [0...n]}$ such that $\mathcal{T}(s_i, a_i) = s_{i+1}$ and $\mathcal{T}(s_n, a_n) = g$. The *solution length* is the number of state-action (s, a) pairs in the solution.[2] A state, s, is said to be *expanded* if $\mathcal{T}(s, a)$ is queried for all valid $a \in A$. We consider *heuristic functions* and *policies* as guiding functions. The heuristic function $h : S \rightarrow \mathbb{R}$ is used to guide a search algorithm, where $h(s)$ is an estimate of the minimal solution length from the given state s. A policy is defined as $\pi : S \times A \rightarrow [0, 1]$ where $\pi(s, a)$ is the probability of taking action a from state s, such that $\sum_{a \in A'} \pi(s, a) = 1$ where A' is the set of all valid actions from state s.

We denote a guided search algorithm by SA_θ where SA is the search algorithm and θ are the parameters of the guiding function. The task of a search algorithm, SA, given a problem instance, P, is to find a solution that either minimizes the solution length or the number of expansions. The specific objective depends on the search algorithms e.g., Levin Tree Search (Orseau and Lelis 2021) minimizes the number of expansions while A* (Hart, Nilsson, and Raphael 1968) can be viewed to find an optimal solution (i.e., minimize the solution length).[3] The expansion budget denotes the maximum number of states that can be expanded by the search algorithm per problem instance. Such a constraint on the number of expansions might be crucial for time sensitive applications. A training set $D = \{P_1, P_2, ..., P_n\}$ is a set of problem instances of a search problem used to train the guiding function. Similarly, a test set $T = \{P'_1, P'_2, ..., P'_n\}$ is a set of problem instances of a search problem that are used to evaluate the performance of the guided search algorithm.

[1] Code is available here: https://github.com/Pi-Star-Lab/TSC-search-problems

[2] We expect all learning methods discussed in the paper to generalize to cases where the cost of taking an action is not the same.

[3] A* is optimal under the assumption of an admissible heuristic function.

Algorithm 1 Bootstrap Learning Method

1: **Input:** maximum expansion budget b, initial expansion budget t, guided search algorithm SA_θ, test set T, training set D
2: **Output:** SA_{θ^*} ▷ SA with updated parameters of guiding function
3: **while** SA_θ cannot solve 100% of T within b **do**
4: results $\leftarrow \text{SA}_\theta$.search($D$, t)
5: $\theta \leftarrow \text{SA}_\theta$.update(results.solutions)
6: **if** number of unsolved instances $\leq$ threshold **then**
7: $t \leftarrow t \times 2$
8: **end if**
9: $D \leftarrow$ unsolved
10: **end while**

2.2 Problem Definition

The task of a learning algorithm L is to be able to efficiently (in terms of training time) solve 100% of the test set. That is, given a guided search algorithm SA_θ, test set, and an expansion budget, solve 100% of the test set within the given expansion budget while being as efficient as possible in terms of training time. Our paper focuses on developing a learning algorithm that achieves better efficiency.

2.3 Bootstrap Learning Method (BL)

A generalized version of the BL is described in the Algorithm 1. BL starts with a fixed training set, a guided search algorithm (SA_θ), an initial expansion budget (per instance), and a maximum expansion budget on the number of expansions (per instance). It attempts to solve the training set within the expansion budget (initially assigned to be the initial expansion budget). It uses the solutions of the instances that could be solved within the expansion budget to update the parameters of the guiding function θ. If the guided search algorithm is not able to solve a certain number of problem instances (threshold), then it doubles the expansion budget. Finally, the BL method removes the solved instances from the training set. This process is continued in a loop until the guided search algorithm can solve 100% of the test set.

TSC differs from BL in two ways (in addition to using a teacher-student curriculum).

1. Instead of requiring fixed training data, TSC generates new training instances at each iteration. Consequently, TSC does not store the unsolved problems (unlike line 9 in Algorithm 1).
2. As the curriculum is generated by TSC, it does not require changing expansion budgets to filter out difficult problem instances. Therefore, TSC uses the same expansion budget for each run of the search algorithm and thus eliminates the need for lines 6, 7, and 8 in Algorithm 1. This also reduces a hyperparameter that requires tuning.

2.4 Enhanced Random Walk (RW+)

Algorithm 2 in (Arfaee, Zilles, and Holte 2010), referred to as RW, was proposed as a solution when BL fails to solve a minimum threshold of problem instances from the training set. RW generates problem instances by performing random walks from the goal state so that the new instances can be solved with BL. RW increases the length of the random walks at each iteration by performing two steps: (1) a breadth first search from the goal state until the expansion budget is exceeded, and (2) Performing 5,000 random walks from the goal until a state that is not visited in (1) is encountered. We empirically observed that such increments led to unstable learning i.e., for some domains, it could solve all instances, for others, it could not solve 100% of the test set (see Subsection 5.6 for details). We suspect this occurs as RW does not take into account the current performance of the guided search algorithm when increasing the length of random walks which in turn leads to instability. Note that RW is designed to produce training instances of progressive difficulties, however, it does not specifically target computational efficiency (which is our main objective as proposed in Subsection 2.2). Consequently, we adapt RW to achieve improved training times by presenting a computationally lighter and stable version denoted RW+. RW+ employs a simple strategy to increase the length of random walks by 1 if the guided search algorithm can solve 75% (a tuned hyperparameter) of the problem instances generated. Note that the increase by 1 might still lead to slower learning for some search problems. This is because, an increase by 1 might still generate easier samples, where the guided search algorithm might benefit more from difficult instances, thus wasting computing resources. This drawback is addressed by our novel TSC method (presented in Section 4). Furthermore, at each iteration, RW+ only generates a small number of problem instances (equal to the batch size used for the guided search algorithms). This is because we observed faster learning initially, thus solving a fewer number of easy instances saves training time. As the curriculum is generated by RW+, the expansion budget does not need to be changed (unlike BL) reducing a hyperparameter. RW+ is outlined in Algorithm 2.

Algorithm 2 Enhanced Random Walk (RW+)

Input: maximum expansion budget b, guided search algorithm SA_θ, test set T
Output: SA_{θ^*} ▷ SA with updated parameters of guiding function
len $\leftarrow 4$ ▷ a hyper parameter
while SA_θ cannot solve 100% of T within b **do**
    $D \leftarrow$ random_walk(len)
    results $\leftarrow \text{SA}_\theta$.search($D$, b)
    $\theta \leftarrow \text{SA}_\theta$.update(results.solutions)
    if SA_θ can solve 75% of the problems in D **then**
        len += 1
    end if
end while

3 Related Work

First, we discuss the recent advances in learning guiding functions for guided search algorithms, concluding with a brief list of guided search algorithms that can be used with our TSC approach. Following, we discuss curriculum generation and learning methods while discussing their applicability to online guided search algorithms.

3.1 State-Space Search and Machine Learning

Bootstrap Learning of heuristic functions (BLH) (Arfaee, Zilles, and Holte 2010) was one of the initial approaches, that proposed learning a heuristic function for the A* algorithm (Hart, Nilsson, and Raphael 1968). Following, McAleer et al. (2018) proposed the DeepCube method that learns the heuristic function with reinforcement learning to solve Rubik's cube instances. They proposed generating the training problem instances from previously solved instances and using the temporal difference method for learning. The more advanced DeepCubeA (Agostinelli et al. 2019) builds upon DeepCube. DeepCubeA performs temporal difference updates for all the states along the random walk performed from goal states. Recent work also investigates learning a heuristic function for classical planning algorithms (Shen, Trevizan, and Thiébaux 2020; Gehring et al. 2022).

Orseau et al. (2018) proposed a novel search algorithm that is guided by a trained policy represented by a neural network referred to as Levin Tree Search (LTS). Orseau and Lelis (2021) combined both, policy and heuristic function learning, resulting in a new algorithm referred to as Policy-guided Heuristic Search (PHS).

BLH, LTS, PHS, and PHS* (a variant of PHS proposed by Orseau and Lelis (2021)) are online learning search algorithms and use the bootstrap method (Arfaee, Zilles, and Holte 2010) as their learning procedure. In this work, we improve upon the Bootstrap Learning method by introducing a curriculum generation method.

3.2 Curriculum Generation

Curriculum generation and learning is a widely studied topic in the field of reinforcement learning (Narvekar et al. 2020). Such approaches can be divided into several categories. The teacher-guided curriculum involves a teacher who guides the student or the reinforcement learning algorithm by increasing the difficulty of the problem with the improving performance of the student (Matiisen et al. 2019; Portelas et al. 2020). Our TSC method is motivated by such methods, where we order the tasks based on the number of random steps taken from the goal as a proxy for an instance difficulty. Another class of algorithms involves generating achievable sub-goals for the student. Florensa et al. (2018) proposed one such approach that involves a Goal Generative Adversarial Network. Other classes include dividing the tasks into specific skills and learning them individually (Jabri et al. 2019) and self-play based curriculum generation (Sukhbaatar et al. 2018). However, these methods are not desirable for online search algorithms, as most

of such methods assume no access to the transition function, and thus use relatively complex mechanisms to generate curricula (e.g., using a neural network to generate initial states). Consequently, curriculum generation with such methods is time inefficient in problems where the transition function is assumed to be known. Recent work has highlighted the advantages of using curriculum generation methods for combinatorial optimization (Lisicki, Afkanpour, and Taylor 2020; Iklassov et al. 2023). Lelis et al. (2022) used the Bootstrap Learning method to generate a curriculum for humans in the 'Sokoban' domain where random walks from goal were not meaningful. Ferber et al. (2022) propose two enhancements to the Bootstrap Learning method within the FDR planning framework where (variant 1, Bootstrapping a Goal-Distance Estimator) introduces a unique approach for performing random walks in partial assignment spaces, and (variant 2, Bootstrapping a Search-Space-Size Estimator) replaces the common heuristic value target (cost-to-goal) with a search-space size estimation. These enhancements are orthogonal to the learning method and can thus potentially be incorporated into our teacher-student learning method in the future. Torralba, Seipp, and Sievers (2021) introduced "Autoscale" tool to generate problem instances of progressive difficulty for evaluating planners. Although Torralba, Seipp, and Sievers (2021) and our paper attempt to generate problem instances of progressive difficulty, the primary objective vastly differs. Torralba, Seipp, and Sievers (2021) generate progressively difficult instances so that all the planners can be evaluated. Our objective, by contrast, targets generating a curriculum for a student (a specific search algorithm) enabling faster training.

4 Teacher-Student Curriculum

In this section, we describe our Teacher-Student Curriculum (TSC) method to address the shortcomings of BL and RW/RW+. First, we describe the TSC method procedurally. Second, we describe and justify a suitable optimization objective for the teacher in TSC. Next, we provide details on the choice of optimizer for the teacher. Finally, we discuss an issue of non-stationarity with the teacher due to improving student's performance and provide a workaround.

4.1 TSC Method

Our TSC approach for training a guiding function is outlined in Algorithm 3. TSC follows an iterative loop similar to BL and RW+. First, the teacher is initialized (Line 3). At each iteration, a batch of integers, representing the difficulty of the training problem instances, is sampled from the teacher (Line 5). Following (Arfaee, Zilles, and Holte 2010; Orseau and Lelis 2021), we use the number of random steps taken from the goal as an estimate of the difficulty of the problem. Problem instances are generated by performing random walks from the goal state; the final state in the random walk is stored in a problem instance buffer D (Line 6). The student (guided search algorithm, SA_θ) is used to solve all problem instances in D (Line 7). The solution paths of the solved problems from D are used as training data to update the parameters of the parameterized guiding function

Algorithm 3 Teacher Student Curriculum (TSC)

1: **Input:** maximum expansion budget b, guided search algorithm SA_θ, test set T
2: **Output:** $\text{SA}_{\theta*}$ ▷ SA with updated parameters of guiding function
3: teacher ← Initialize Teacher algorithm
4: **while** SA_θ cannot solve 100% of T within b **do**
5: random_steps ← teacher.sample()
6: D ← random_walk(random_steps)
7: results ← SA_θ.search(D, b)
8: θ ← SA_θ.update(results.solutions)
9: $costs$ ← Follow Equation 1
10: update teacher using $costs$
11: **end while**

(Line 8). Finally, the teacher is updated based on the performance of SA_θ (the student) (Lines 9 and 10). The details on the teacher's implementation are discussed in Subsection 4.3 and Subsection 4.4. This process continues until SA_θ can solve all problems in T (Line 4).

4.2 Teacher's Objective

As briefly discussed in Section 1, the objective of the teacher is to generate problem instances such that they are easy enough to be solved within the expansion budget, while being difficult enough so that the training data is valuable for achieving a better guiding function. To achieve this, we propose to use Equation 1 as the cost function for the teacher. The objective of the teacher is to minimize this cost function.

$$C(\text{expansions}) = \begin{cases} -\text{expansions}, & \text{if instance is solved} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

The rationale behind the cost function stems from the observation that the goal of the teacher is to generate the most difficult solvable (by the constrained student) problem instances. Difficult instances usually require the student to expand a larger number of states. Thus, the cost function inversely correlates to the number of expanded states. Furthermore, the cost function assigns the highest cost of 0 (as the number of expansions is a non-negative number) to instances that could not be solved within the expansion budget as they cannot be used as training data. Consequently, the teacher attempts to produce instances requiring the maximal number of expansions that is still below the budget b. Solving such problem instances of intermediate difficulty improves efficiency in training time by providing valuable training data while exploring difficult problems.

4.3 CMA-ES as Teacher

We represent the teacher as a univariate Gaussian distribution which is parameterized by initial mean μ_0 and initial standard deviation σ_0. To update these parameters, we use the Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) (Hansen 2006) which is a widely used black-box optimizer. CMA-ES is an iterative evolutionary strategy

where at each iteration i we sample a batch of real numbers from the Gaussian distribution. These numbers denote the length of random walks (after rounding) in our case. Problem instances are generated by performing backward random walks. These instances are evaluated using SA_θ and the results (number of expansions) are used to calculate the cost incurred by the teacher (Equation 1). A fraction of the batch that are best-performing numbers (based on the cost) are referred to as the "fittest". The fittest numbers of the iteration are used to update the μ_i and σ_i of the Gaussian distribution. Specifically, the mean is updated with weighted averaging of fittest numbers and the standard deviation is updated by using 'rank-μ-updates' and 'cumulation' (see Equation 3 and 22 in Hansen (2006) for details). CMA-ES optimization is considered to be converged once the standard deviation drops below a certain threshold value. Empirically, CMA-ES was shown to converge on a wide variety of optimization objectives (Hansen et al. 2010). We choose CMA-ES as it does not require tedious hyperparameter tuning. In addition, the choice of using CMA-ES updates is motivated by the results presented by Hansen et al. (2010) where CMA-ES and its variants were shown to outperform other methods on complex non-convex functions. The initial mean and the initial standard deviation ($\mu_0,\ \sigma_0$) are hyperparameters provided by the user. Typically, the values of the initial mean should be low, as problem instances generated during initial iterations should be easy for the guided search algorithms. On the other hand, σ_0 should be high enough that it encourages exploration of various problem difficulties while being not very high to avoid sampling of problem instances that are too difficult early in the learning process.

4.4 Addressing Non-Stationarity

One problem with implementing the teacher as a CMA-ES optimizer is the fact that it was designed to optimize stationary objective functions, i.e., functions that do not change over time. However, the objective of the teacher is a non-stationary function as the guiding function is updated at each iteration, and the performance of the student changes with time. To address this issue, we propose to reset all parameters of CMA-ES, apart from the mean, if the standard deviation is reduced below a small threshold. This enables a prematurely converged teacher to explore again by generating more difficult problem instances once the guiding function is good enough and thus, avoid premature convergence.

5 Experiments

In this section, we first describe the experimental setup that includes domain descriptions, baseline learning methods, and guided search algorithms. Then, we show a comparison of the learning methods with respect to the training time required to reach a point where all of the instances in the test set can be solved. This is the main result as improving computational efficiency is the primary objective of the paper. Following, we show a comparison in terms of solution length and number of expansions of the guided search algorithms after training the guiding function for a fixed amount of time. These results highlight the advantages of RW+ and TSC over BL and RW. Next, we present a study of how these learning methods scale for harder search problems. This study highlights the limitations of RW+ with the increasing complexity of search problems.

5.1 Domains

We use 3 representative benchmark domains namely Pancake Sorting, Sliding Tile Puzzle, and 4-peg Towers of Hanoi (Pendurkar et al. 2023, 2022; Orseau and Lelis 2021; Arfaee, Zilles, and Holte 2010). Note, we do not include results for Witness Puzzle and Sokoban Puzzle reported by (Orseau and Lelis 2021) as (1) random walks are not applicable to Witness Puzzle.[4] and (2) generating meaningful problem instances for Sokoban with random walks is an open challenge (Bento, Pereira, and Lelis 2019). Extending TSC to such domains is left as future work.

We refer to a Pancake Sorting problem with 'n' pancakes as 'n Pancake' problem. Similarly, we refer to a $n \times n$ Sliding Tile Puzzle as 'n STP', and 4-peg Towers of Hanoi with n disks as 'n TOH'. Details on state encoding for neural networks, and test set generation for the domains are in Appendix A.

5.2 Baselines

We use BL, RW and RW+ as the baseline learning methods.

Bootstrap Learning (BL) (Arfaee, Zilles, and Holte 2010): The training set for all domains was generated by taking random steps from the goal as suggested by Orseau and Lelis (2021) for the STP puzzle. The number of random steps to be taken from the goal was chosen uniformly between 1-100, 50-1000, and 5-1000 for Pancake, STP, and TOH respectively. The size of the training set was set to 2048, 10000, and 2048 for Pancake, STP, and TOH respectively. The initial expansion budget for the BL method for each of the guided search algorithms was set to $1/4$ times the maximum expansion budget for Pancake and STP and $1/8$ for TOH. These parameters were tuned until BL was able to solve at least a single problem instance of the training set within the initial expansion budget.

Random Walk (RW) (Arfaee, Zilles, and Holte 2010): The length was incremented at each iteration following the recommendation by Arfaee, Zilles, and Holte (2010). The initial expansion budget was set to be the same as BL. At each iteration, we sample 32 problem instances - the same as TSC and RW+.

Enhanced Random Walk (RW+): RW+ is implemented as presented in Section 2.4.

5.3 Guided Search Algorithms

We test the learning methods using three guided search algorithms, namely Levin Tree Search, Policy-Guided Heuristic Search, and Weighted A*.

[4]Instead of random walks, Sturtevant (2019) used procedural content generation based techniques.

Levin Tree Search (LTS): LTS is a policy based search algorithm, where the policy is used to guide the search (Orseau et al. 2018). The policy is represented by a neural network.

Policy-Guided Heuristic Search (PHS*): PHS* is one of the variants proposed by Orseau and Lelis (2021), that uses both policy and heuristic function to guide the search. For experiments, we use a two-headed neural network as used by Orseau and Lelis (2021). One head represents the policy and the other one represents the heuristic function.

Weighted A*(WA*): WA* is a widely used representative heuristic search algorithm (Pohl 1973). WA* is guided by a heuristic function which we represent with a neural network. We use a weight of 1.5 for all experiments.

5.4 Evaluation

All learning methods use the same domain-specific test set of 256 problem instances for all cases. We evaluate all methods with two metrics.

- **Metric 1:** We measure the time required for the learning methods to train the guiding function such that the guided search algorithm can solve all instances in the test set. The time includes the wall clock time required to perform the search, update the parameters of the guiding function, and update the parameters of the learning method (e.g., updating the teacher in TSC). The wall clock time required for test set evaluation is not considered part of the reported training time. These experiments were performed with 5 random seeds and the training time is averaged across these runs. We report the standard error following common practices (Orseau and Lelis 2021). Note, we do not perform a specific statistical test (which requires extra assumptions) and usually require a higher number of seeds (e.g., ≥ 30 seeds) which is computationally expensive in our case.
- **Metric 2:** We run the learning methods for a fixed amount of wall clock time and compare the performance of the guided search algorithm with the guiding function obtained when the time expires on the test set. Specifically, we compare the percentage of test set solved, the average solution length, and the average number of expansions. These experiments were performed with 5 random seeds and the 'best' performing run was reported. The run with the highest percentage of test set solved is said to be the best run. In case of a tie, the run with the lowest number of expansions (amongst the solved instances in the test set) is said to be the best. The best runs are reported as aggregating runs is not meaningful when different problem instances of the test set are solved (Similar to Table 1 in (Orseau and Lelis 2021)).

5.5 Other Experimental Setup Details

Our codebase is built upon the one provided by Orseau and Lelis (2021), thus we share the same hyperparameters unless stated otherwise. The batch size (number of problem instances generated at each iteration) for TSC and RW+ is set to 32 following the recommendation (Orseau and Lelis 2021). The expansion budgets used for Pancake, STP, and TOH are 5000, 6000, and 3000 respectively unless stated otherwise. The 'random_steps' in Algorithm 3 was set to zero if the teacher provided a negative value. For CMA-ES we use the benchmark python package by Shibata (2023). The initial mean of the teacher (Gaussian) was set to 4 for all experiments and the initial standard deviation σ_0 was set to 4 for Pancake and STP and 100 for TOH unless stated otherwise. CMA-ES was restarted if the standard deviation drops below 0.01. The experiments were performed on a single computer with Intel(R) Core(TM) i9-10900X CPU 3.70GHz, with two 8 GiB synchronous (2666MHz) RAM. Linux Mint 20.1 was used as the OS with kernel version 5.4.0-70-generic. Note that only one thread/process was used for one run, and no GPU was used. For details on the neural network architecture, please refer to Appendix A.

5.6 Training Time Comparison

First, we investigate the time required for the learning methods to train the guiding function, as this is the primary task. For this investigation, we continue training the guiding function until the guided search algorithm can solve all problems in the test set and measure the time required for training (i.e., Metric 1). Table 1 shows the training time required across the three domains and the three guided search algorithms. The results show that the BL method performs worst across all domains and guided search algorithms. Specifically, BL is approximately 7.5 to 31 times slower to train as compared to TSC, across the search problems and search algorithms. On the other hand, RW+ is competitive with TSC. RW+ has the edge over TSC with WA* for 4 STP. In all other cases, TSC outperforms RW+, where the best results are obtained for TOH with WA* where TSC is around 4 times faster. Note that the standard error across runs is similar for TSC and RW+ for all domains and guided search algorithms and is significantly lower than the BL method. We suspect the standard error is higher for BL because we resample a new training set at each run, highlighting the importance of selecting appropriate problem instances for training. On the other hand, RW could only solve all instances of the test set for 9 TOH and 16 Pancake with only PHS* algorithm resulting in an unstable performance as opposed to other learning methods. RW is competitive with TSC for 9 TOH. For 16 Pancake and PHS*, RW is 4.9 times slower as compared to TSC.

Given the improvements in training time, we examined whether the improved training time comes at the cost of a lower-quality guiding function. In order to address this question, we run all of the learning methods for a fixed amount of time and evaluate the guided search algorithms on the test set after the time expires (Metric 2).[5] Table 2 shows the average solution length and average number of expansions performed for all the learning methods and the guided search algorithms for 4 STP. The results suggest that TSC outperforms BL with PHS*and LTS algorithms where the results are comparable to BL when WA* is used as the guided search algorithm. Further, we observe that TSC results in

[5]we exclude RW as it could not solve all instances of the test set for most of the domains, guided search algorithms

Domain	Method	Training time required to solve all problems in test set (in seconds)					
		PHS*		LTS		WA*	
16 Pancake	TSC	3,366	(1,079)	2,607	(218)	7,882	(677)
	RW+	3,757	(1,975)	3,761	(421)	8,451	(1,156)
	RW	16,496	(6,515)	NA	(NA)	NA	(NA)
	BL	38,417	(5,245)	47,901	(9,550)	94,364	(14,712)
4 STP	TSC	3,871	(963)	5,349	(938)	3,745	(1,151)
	RW+	5,929	(787)	9,449	(2,534)	3,513	(1,166)
	RW	NA	(NA)	NA	(NA)	NA	(NA)
	BL	110,629	(32,155)	164,100	(71,406)	96,605	(33,593)
9 TOH	TSC	1,876	(106)	1,406	(60)	8,456	(4,074)
	RW+	2,460	(180)	1,647	(464)	34,050	(20,007)
	RW	1,903	(37)	1,453	(85)	9,720	(628)
	BL	17,860	(649)	10,747	(1,860)	136,886	(NA)*

Table 1: A performance comparison of the learning methods across three domains and three guided search algorithms. The training time reported is averaged across 5 random seeds. The value to the left (outside the bracket) is the mean and the value to the right (inside the bracket) is the standard error across the runs. BL with WA* for 9 TOH could not solve all instances in the test set for 4/5 runs (marked with '*').

Method	PHS*		LTS		WA*	
	Length	Expansions	Length	Expansions	Length	Expansions
TSC	65.98	899.8	65.80	896.2	59.27	918.4
RW+	79.91	1,104.4	68.82	934.9	59.27	918.4
BL	71.14	978.7	84.00	1,153.0	60.41	934.6

Table 2: Performance of guided search algorithms for 4 STP on the test set after training for a time of 50,000 seconds. The guided search algorithms solved all problems in the test set for all of the cases. The Length and the Expansions columns are averaged over solutions of all the instances in the test set. These results are for the best run out of the 5 runs of the experiment following Metric 2.

better performance within the time limit than RW+ for LTS and PHS*. We suspect the worse performance of RW+ and BL in certain cases is because the guiding function has not yet converged. This implies that on training for longer (at the limit), all of the learning methods might result in a similar performance for the guided search algorithms. This further supports our claim that TSC is a faster method as it results in quicker convergence. Note, similar trends were observed for 16 Pancake and 9 TOH and the results are reported in Appendix B.

5.7 Scaling to Harder Search Problems

In this section, we study how well the performance of the learning methods scales to harder search problems. For this study, we use Metric 2, i.e., we run the learning method for a fixed time and then evaluate the guided search algorithms on the test set. The results on 5 STP, 24 Pancake, and 12 TOH are shown in Table 3. Note that, we exclude BL and RW from this study as BL could not solve any instances from the test set within the time budget and RW failed to solve all instances of the test set for several domains, guided search algorithms (see Subsection 5.6). From the results we see in cases where both TSC and RW+ can solve 100% of the test set, TSC returns better solutions in terms of either the average length of solution, number of expansions, or both. In cases where TSC or RW+ could not solve 100% of the test set, we see that TSC can solve more instances from the test set (e.g., 12 TOH where TSC solves a higher fraction of problem instances from the test set across all guided search algorithms). Note that, in such cases, the solution quality cannot be directly compared as TSC could be solving difficult problem instances resulting in higher solution lengths and number of expansions (or vice versa). WA* on 24 Pancake is the exception where both TSC and RW+ fail to solve any instance of the test set. The biggest advantage for TSC is observed for LTS and 5 STP, where TSC can solve all problem instances in the test set as opposed to RW+ which could not even solve a single instance. Note, these results align with the results presented in (Orseau and Lelis 2021) where LTS could solve 0.9% of the test set by training it for 6 times longer and without a fixed expansion budget with BL (on the test set).

These results show a wider gap between the performance of TSC and RW+ as compared to the one discussed in Sub-

Method	24 Pancake								
	PHS*			LTS			WA*		
	Solved	Length	Expansions	Solved	Length	Expansions	Solved	Length	Expansions
TSC	100%	115.96	234.46	100%	122.19	245.5	0%	-	-
RW+	100%	136.45	276.77	100%	135.69	273.0	0%	-	-
Method	**5 STP**								
	PHS*			LTS			WA*		
	Solved	Length	Expansions	Solved	Length	Expansions	Solved	Length	Expansions
TSC	100%	251.62	3,231.7	100%	173.91	2,362.7	100%	150.03	2,936.1
RW+	100%	301.27	3,902.3	0%	-	-	100%	161.20	2,880.0
Method	**12 TOH**								
	PHS*			LTS			WA*		
	Solved	Length	Expansions	Solved	Length	Expansions	Solved	Length	Expansions
TSC	37%	64.25	2,075.9	75%	93.05	3,756.6	11%	43.75	2,529.9
RW+	30%	72.89	1,337.2	48%	76.70	4,227.5	3%	37.75	6,390.5

Table 3: A comparison of the performance of the learning methods for harder search problems. The Length and the Expansions columns are averaged over solutions of all the instances that could be solved from the test set. Hence, if 100% instances are not solved, appropriate comparison with respect to Lengths and Expansions is not feasible. The budget used for Pancake, STP, and TOH is 25,000, 50,000, and 20,000 respectively. The time limit used for Pancake, STP, and TOH is 100,000, 200,000, and 100,000 respectively. These results are for the best run out of the 5 runs of the experiment.

section 5.6. This is because, higher lengths of random walks might result in loops (i.e., revisiting previously seen states), thus increasing the length by 1 at each iteration might not be sufficient. The results and understandings suggest that TSC might scale better with the increasing complexity of the search problems as opposed to RW+.

6 Summary

This paper tackles the problem of generating a curriculum of problem instances for training guided search algorithms to improve computational efficiency. First, we present an enhanced version of an existing method referred to as RW+ that specifically attempts to improve the computational efficiency and stability of the RW method. Following, we propose a curriculum based learning method referred to as Teacher Student Curriculum (TSC). TSC consists of two components (1) teacher: the task of the teacher is to generate problem instances of "correct difficulty", and (2) student: the student is any search algorithm guided by a parameterized function. Experiments performed across three search problems and three guided search algorithms suggest that TSC is 7.5 to 31 times faster than the state-of-the-art Bootstrap Learning method. We also present results for harder search problems that highlight the scalability of TSC over RW+. Future work will explore extensions to TSC so that it can be applied to domains where backward random walks cannot be used (e.g., the Sokoban and Witness domains).

Acknowledgements

This research was partially supported by Canada's NSERC and the CIFAR AI Chairs program, and enabled in part by support provided by the Digital Research Alliance of Canada. A part of the research has taken place in the PiStar AI and Optimization Lab at Texas A&M University. PiStar is supported in part by NSF (IIS-2238979) and BSF (2022191).

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's Cube with Deep Reinforcement Learning and Search. *Nature Machine Intelligence*, 1(8).

Agostinelli, F.; McAleer, S.; Shmakov, A.; Fox, R.; Valtorta, M.; Srivastava, B.; and Baldi, P. 2021. Obtaining Approximately Admissible Heuristic Functions through Deep Reinforcement Learning and A* Search. *Bridging the Gap between AI Planning and Reinforcement Learning workshop at International Conference on Automated Planning and Scheduling*.

Arfaee, S. J.; Zilles, S.; and Holte, R. 2010. Bootstrap learning of heuristic functions. In *Annual Symposium on Combinatorial Search*, 52–60.

Bento, D. S.; Pereira, A. G.; and Lelis, L. H. 2019. Procedural generation of initial states of sokoban. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 4651–4657.

Ferber, P.; Geißer, F.; Trevizan, F.; Helmert, M.; and Hoffmann, J. 2022. Neural network heuristic functions for classi-

cal planning: Bootstrapping and comparison to other methods. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 583–587.

Florensa, C.; Held, D.; Geng, X.; and Abbeel, P. 2018. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, 1515–1528. PMLR.

Gehring, C.; Asai, M.; Chitnis, R.; Silver, T.; Kaelbling, L.; Sohrabi, S.; and Katz, M. 2022. Reinforcement learning for classical planning: Viewing heuristics as dense reward generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 588–596.

Hansen, N. 2006. The CMA evolution strategy: a comparing review. *Towards a new evolutionary computation: Advances in the estimation of distribution algorithms*, 75–102.

Hansen, N.; Auger, A.; Ros, R.; Finck, S.; and Pošík, P. 2010. Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In *Proceedings of the conference companion on Genetic and evolutionary computation*, 1689–1696.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Iklassov, Z.; Medvedev, D.; De Retana, R. S. O.; and Takac, M. 2023. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, 5350–5358.

Jabri, A.; Hsu, K.; Gupta, A.; Eysenbach, B.; Levine, S.; and Finn, C. 2019. Unsupervised curricula for visual meta-reinforcement learning. *Advances in Neural Information Processing Systems*, 32: 10519–10531.

Lelis, L. H. S.; ao G. G. V. Nova, J.; Chen, E.; Sturtevant, N. R.; Epp, C. D.; and Bowling, M. 2022. Learning Curricula for Humans: An Empirical Study with Puzzles from The Witness. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Li, T.; Chen, R.; Mavrin, B.; Sturtevant, N. R.; Nadav, D.; and Felner, A. 2022. Optimal Search with Neural Networks: Challenges and Approaches. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 109–117.

Lisicki, M.; Afkanpour, A.; and Taylor, G. W. 2020. Evaluating Curriculum Learning Strategies in Neural Combinatorial Optimization. In *Learning Meets Combinatorial Algorithms Workshop at Neural Information Processing Systems*.

Matiisen, T.; Oliver, A.; Cohen, T.; and Schulman, J. 2019. Teacher–student curriculum learning. *IEEE Transactions on Neural Networks and Learning Systems*, 31(9): 3732–3740.

McAleer, S.; Agostinelli, F.; Shmakov, A.; and Baldi, P. 2018. Solving the Rubik's Cube with Approximate Policy Iteration. In *International Conference on Learning Representations*.

Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M. E.; and Stone, P. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of Machine Learning Research*, 21(181): 1–50.

Orseau, L.; Lelis, L.; Lattimore, T.; and Weber, T. 2018. Single-agent policy tree search with guarantees. *Advances in Neural Information Processing Systems*.

Orseau, L.; and Lelis, L. H. 2021. Policy-guided heuristic search with guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 12382–12390.

Pendurkar, S.; Huang, T.; Juba, B.; Zhang, J.; Koenig, S.; and Sharon, G. 2023. The (Un)Scalability of Informed Heuristic Function Estimation in NP-Hard Search Problems. *Transactions on Machine Learning Research*.

Pendurkar, S.; Huang, T.; Koenig, S.; and Sharon, G. 2022. A discussion on the scalability of heuristic approximators. In *Proceedings of the International Symposium on Combinatorial Search*, volume 15, 311–313.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, 12–17.

Portelas, R.; Colas, C.; Hofmann, K.; and Oudeyer, P.-Y. 2020. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, 835–853. PMLR.

Shen, W.; Trevizan, F.; and Thiébaux, S. 2020. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 574–584.

Shibata, M. 2023. Lightweight Covariance Matrix Adaptation Evolution Strategy (CMA-ES) implementation for Python 3. *URL: https://pypi.org/project/cmaes/*.

Sturtevant, N. R. 2019. Exploring EPCG in The Witness. In *Knowledge Extraction from Games (AAAI workshop)*, 58–63.

Sukhbaatar, S.; Lin, Z.; Kostrikov, I.; Synnaeve, G.; Szlam, A.; and Fergus, R. 2018. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In *International Conference on Learning Representations*.

Torralba, A.; Seipp, J.; and Sievers, S. 2021. Automatic instance generation for classical planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 376–384.