Structure and randomness in planning and reinforcement learning

Piotr Kozakowski[*]
University of Warsaw
p.kozakowski@mimuw.edu.pl

Piotr Januszewski[*]
Gdansk University of Technology
& University of Warsaw
piotr.januszewski@pg.edu.pl

Konrad Czechowski[*]
University of Warsaw
k.czechowski@mimuw.edu.pl

Łukasz Kuciński
Polish Academy of Sciences
lkucinski@impan.pl

Piotr Miłoś
Polish Academy of Sciences
pmilos@impan.pl

Abstract

Planning in large state spaces inevitably needs to balance depth and breadth of the search. It has a crucial impact on planners performance and most manage this interplay implicitly. We present a novel method *Shoot Tree Search (STS)*, which makes it possible to control this trade-off more explicitly. Our algorithm can be understood as an interpolation between two celebrated search mechanisms: MCTS and random shooting. It also lets the user control the bias-variance trade-off, akin to $TD(n)$, but in the tree search context.

In experiments on challenging domains, we show that STS can get the best of both worlds consistently achieving higher scores.

1 Introduction

Classically, reinforcement learning is split into model-free and model-based methods. Each of these approaches has its strengths and weaknesses: the former often achieves state-of-the-art performance, while the latter holds the promise of better sample efficiency and adaptability to new situations. Interestingly, in both paradigms, there exists a non-trivial interplay between structure and randomness.

In the model-free approach, Temporal Difference (TD) prediction leverages the structure of function approximators, while Monte Carlo (MC) prediction relies on random rollouts. Model-based methods often employ planning, which counterfactually evaluates future scenarios. The design of a planner can lean either towards randomness, with random rollouts used for state evaluation (e.g. random shooting), or towards structure, where a data-structure, typically a tree or a graph, forms a backbone of the search (e.g. Monte Carlo Tree Search).

We present a novel method: Shoot Tree Search (STS). The development of the algorithm was motivated by the aforementioned observations concerning structure, randomness, and dilemma between breadth and depth of the search. It lets the user control the depth and breadth of the search more explicitly and can be viewed as a bias-variance control method. STS itself can be understood as an interpolation between MCTS and random shooting. We show experimentally that, on a diverse set

[*]equal contribution, random order of authors

of environments, STS can get the best of both worlds. We also provide some toy environments, to get an insight into why STS can be expected to perform well. The critical element of STS, *multi-step expansion*, can be easily implemented on top of many algorithms from the MCTS family. As such, it can be viewed as one of the extensions in the MCTS toolbox.

2 Background and related work

The introduction to reinforcement learning can be found in Sutton and Barto [16]. In contemporary research, the line between model-free and model-based methods is often blurred. An early example is Guo et al. [6], where MCTS plays the role of an 'expert' in DAgger (Ross and Bagnell [12]), a policy learning algorithm. In the series of papers Silver et al. [14, 15], culminating in AlphaZero, the authors developed a system combining elements of combinatorial algorithms and reinforcement learning methods that master the game of Go (and others). Similar ideas were also studied in Anthony et al. [1]. In Miłoś et al. [10], planning and model-free learning were brought together to solve combinatorial environments. As most of these works, we use the model-based reinforcement learning paradigm, in which the agent has access to a true model of the environment.

3 Methods

A Generic Planner, presented in Algorithm 1, gives a unified view on all methods analyzed in the paper: Random Shooting, MCTS and, our novel approach, STS. By a suitable choice of functions SELECT, EXPAND, UPDATE and CHOOSE_ACTION, we can recover each of these methods.

Typically, a planner is a part of a training process, see Algorithm 2. In a positive feedback loop, the planner improves the quality of data used for training of the value function V_θ and a policy π_ϕ. Conversely, the policy and value function might further improve planning. Implementation details of Algorithm 2 are provided in Appendix A.1.

In the Appendixes A.4 and A.5 we give a detailed description of the Random Shooting and MCTS planning methods. Below we present our novel planner.

Algorithm 1 Generic Planner, defines required constants, variables and objects used in further algorithms

```
Require:  $C$            planning passes
          $H$            planning horizon
          $\gamma$       discount factor
Use:      $N(s,a)$       visit count
          $W(s,a)$       total action-value
          $Q(s,a)$       mean action-value
          $\mathbf{V}_\theta$   value function
          $\pi_\phi$     policy
          $model$        environment simulator
# Initialize $N, W, Q$ to zero
function PLANNER(state)
    for $1 \dots C$ do
        path, leaf ← SELECT(state)
        rollout, leaf ← EXPAND(leaf)
        UPDATE(path, rollout, leaf)
    return CHOOSE_ACTION(state)
```

Algorithm 2 Training loop, additionally requires environment env

```
# Initialize parameters of $\mathbf{V}_\theta, \pi_\phi$
# Initialize $replay\_buffer$
repeat
    episode ← COLLECT_EPISODE
    $replay\_buffer$.ADD(episode)
    $B \leftarrow replay\_buffer$.BATCH
    Update $\mathbf{V}_\theta, \pi_\phi$ using $B$ and SGD
until convergence
function COLLECT_EPISODE
    $s \leftarrow env$.RESET
    episode ← []
    repeat
        $a \leftarrow$ PLANNER(s)
        $s', r \leftarrow env$.STEP($a$)
        episode.APPEND($(s, a, r, s')$)
        $s \leftarrow s'$
    until episode is done
    return CALCULATE_TARGET(episode)
```

Shoot Tree Search Shoot Tree Search (STS) extends MCTS in a novel way, by redesigning the expansion phase, see Algorithm 3. Given a leaf and a planning horizon H the method expands H consecutive vertices starting from the leaf. Each new node is chosen according to the in-tree policy and is added to the search tree. Note a crucial difference between STS and vanilla MCTS using random rollouts: in contrast to the latter, STS adds visited nodes to the tree, so the explored paths can easily be branched out during later planning passes. We call this mechanism *multi-step expansion*.

Algorithm 3 Shoot Tree Search

```
function EXPAND(leaf)
    $s \leftarrow$ leaf
    rollout $\leftarrow$ []
    for $1 \dots H$ do
        MCTS.EXPAND($s$)
        $a \leftarrow$ CHOOSE_ACTION($s$)
        $s', r \leftarrow tree[s][a]$
        rollout.APPEND($(s, a, r)$)
        $s \leftarrow s'$
    return rollout, $s$

function SELECT(state)
    The same as in Algorithm 7.

function CHOOSE_ACTION($s$)
    The same as in Algorithm 7.

function UPDATE(path, rollout, leaf)
    $s' \leftarrow$ leaf
    $c \leftarrow 1$
    quality $\leftarrow 0$
    for $s, a, r \leftarrow$ reversed(path + rollout) do
        if $s' \in$ path then
            v $\leftarrow 0$
        else
            v $\leftarrow \mathbf{V}_\theta(s')$
            $c \leftarrow c + 1$
        quality $\leftarrow c * r + \gamma *$ (quality + v)
        $W(s,a) \leftarrow W(s,a) +$ quality
        $N(s,a) \leftarrow N(s,a) + c$
        $Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)}$
        $s' \leftarrow s$
```

STS can be viewed as a sophisticated version of Random Shooting applied to MCTS. In this interpretation, STS interpolates between the two methods. We demonstrate empirically that the change introduced by STS is essential to solving challenging RL domains; see Section 4. We note that $H = 1$ corresponds to MCTS.

4 Experiments

We tested the spectrum of algorithms presented in Section 3 on the Sokoban and Google Research Football domains. Those tasks present numerous challenges, which evaluate various properties of planning algorithms. The training details, a list of hyper-parameters, network architectures and *multi-step expansion* analysis are presented in appendices A.1, A.2, A.3 and A.9 respectively.

4.1 Sokoban

Scenario	C	H	S. rate	N_p	N_t	N_g
av. loops	256	1	95.2%	1224	1224	716
	64	4	96.5%	299	1194	830
	16	16	95.7%	114	1822	1333
	4	64	89%	62	3960	1491
no av. loops	256	1	84.5%	1497	1497	376
	32	8	88.4%	185	1483	409
	2	128	65.3%	36	4589	967

Table 1: Comparison on evaluation of MCTS and STS. C, H are parameters in Algorithm 1. S. rate is the ratio of solved boards, $N_p, N_t (= N_p \cdot H), N_g$ are the average number of passes, tree nodes and game states observed until the solution is found. Full table is available in Table 4.

In the first experiment, we evaluated the planning capabilities of STS. We used a pre-trained value function and varied the number of passes C and the depth of multi-step expansion H, such that $H \cdot C$ remains constant. In Table 1 we present quantities (N_p, N_t, N_g), which measure planning costs. In two presented scenarios, there is a sweet spot for the choice of H. For this choice, the number of tree nodes, N_t, which is the most important metric, is the smallest. Interestingly, we observe an increase in the solved rate. This may possibly be explained by the fact that the number of distinct visited game states, N_g, grows. This suggests that STS explores more aggressively and efficiently.

In the second line of experiments, we analyzed the training performance (see Algorithm 2). For MCTS we used $C = 50$ passes per step, while for STS we considered $C = 10$ passes with multi-step expansion $H = 5$. The learning curve for STS dominates the learning curve for MCTS, which persists throughout training, see Figure 1. Since the difficulty of Sokoban levels increases progressively, the achieved improvement is substantial, even though in absolute terms, it may seem small.

Random Shooting methods perform poorly for Sokoban: we evaluated Bandit Shooting (Algorithm 5), which struggled to exceed 5% solved rate.

4.2 Google Research Football

For a description of Google Research Football see Appendix A.8.

STS and MCTS STS achieves state-of-the-art results on the GRF Academy and significantly outperforms other methods. For STS we used $C = 30$ passes with $H = 10$ and for MCTS we set corresponding $C = 300$.

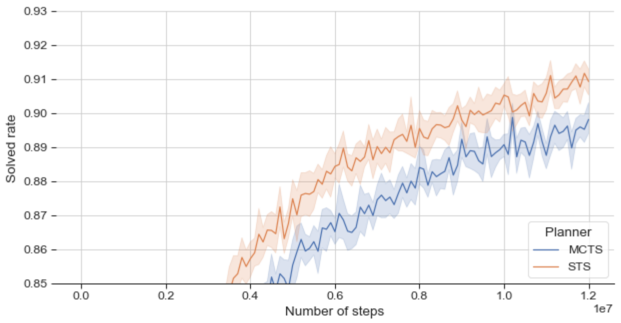


Figure 1: Learning curve for Sokoban domain for limited interval of values on the y axis. The results are averaged over 10 runs, shaded areas shows 95% confidence intervals. The x axis is the number of collected samples. See Figure 4 for the full version.

STS Conv. completely solves 8 out of 11 academy environments, see full results in Table 5, and is the best or close to the best on the remaining 3. One can observe that in Corner, Counterattack easy and hard, Pass and shoot with keeper, Run to score with keeper, and Single goal vs. lazy academies the difference between STS and MCTS, as well as most of the other methods, is substantial. See Table 2 for exact results. We stress that STS Conv. easily beats any other method in one-to-one comparison across all academies. STS MLP achieves a close second place. These results provide further evidence that STS gives a boost in environments requiring long-horizon planning. This stands in sharp contrast with MCTS, which was not able to achieve impressive results in the considered time budget. In the STS experiments we used approx. 0.8M training samples (median). More details can be found in Appendix A.8, including ablations.

Method		3 vs. 1 with keeper	Corner	Counterattack easy	Counterattack hard	Pass and shoot with keeper	Run pass and shoot with keeper	Run to score	Run to score with keeper	Single goal versus lazy
	PPO	0.90	0.10	0.70	0.65	0.65	0.90	0.90	**1.00**	0.90
Random Shooting	flat	0.10	0.00	0.05	0.10	0.05	0.10	0.00	0.00	0.00
	PPO	0.45	0.10	0.10	0.30	0.25	0.80	0.80	0.20	0.30
	MLP	0.90	**0.87**	0.80	0.73	0.87	0.70	0.90	0.37	0.67
Bandit Shooting	flat	0.20	0.10	0.00	0.00	0.05	0.05	0.05	0.00	0.00
	PPO	**1.00**	0.05	0.95	0.80	0.55	**1.00**	0.85	0.45	0.60
	MLP	0.87	0.47	0.73	0.60	0.90	0.80	0.93	**1.00**	0.60
	Conv.	0.97	0.41	0.81	0.44	0.94	0.69	**1.00**	0.91	0.00
MCTS Conv.		0.81	0.50	0.31	0.31	0.45	0.89	0.70	0.00	0.00
STS	MLP	**1.00**	0.78	**1.00**	0.97	0.94	0.97	**1.00**	0.94	0.94
	Conv.	**1.00**	0.81	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.97	**0.97**

Table 2: Summary of selected algorithms' performance on GRF. Entries correspond to solved rates across at leat 20 episodes per environment and these are medians across at least 3 seeds. PPO results come from Kurach et al. [8]. Results for the whole GRF Academy are presented in Table 5.

5 Conclusions

In this paper, we introduced a new algorithm, Shoot Tree Search. STS aims to explicitly address the dilemma between depth and breadth search in large state spaces. Having empirically verified the efficiency of this extension in many challenging scenarios, we argue that it could be included in a standard MCTS toolbox.

References

[1] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *NIPS*, 2017.

[2] 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.

[3] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[4] Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

[5] Matthew L. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 2001. ISSN 10769757. doi: 10.1613/jair.820.

[6] Xiaoxiao Guo, Satinder P. Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *NIPS*, 2014.

[7] Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Tobias Pfaff, Theophane Weber, Lars Buesing, and Peter W. Battaglia. Combining q-learning and search with amortized value estimates. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=SkeAaJrKDS`.

[8] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180*, 2019.

[9] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, and Sylvain Gelly. Google research football. `https://github.com/google-research/football`, 2019.

[10] Piotr Miłoś, Łukasz Kuciński, Konrad Czechowski, Piotr Kozakowski, and Maciek Klimek. Uncertainty-sensitive learning and planning with ensembles. *arXiv preprint arXiv:1912.09996*, 2019.

[11] Sébastien Racanière, Theophane Weber, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. In *NIPS*, 2017.

[12] Stéphane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979, 2014. URL `http://arxiv.org/abs/1406.5979`.

[13] Brian Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 2002. ISSN 00043702. doi: 10.1016/S0004-3702(01)00166-7.

[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 2017.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 1144:1140–1144, 2018.

[16] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

A.1 Training details

We provide the code of our methods and hyper-parameters configuration files in `https://github.com/shoot-tree-search/sts`.

The training loop follows the logic of Algorithm 2. We use a distributed setup with 30 workers and a replay buffer of size 30000. We perform 1000 optimizer updates on batches of transitions whenever all workers collect and store one full episode. During batch sampling, we ensured an equal amount of examples from solved and unsolved episodes. In GRF and Sokoban experiments, each episode was limited to 100 and 200 time steps, respectively.

A value function approximator, $\mathbf{V}_\theta$, is trained via the MSE loss and "reward-to-go" targets $\sum_{i=t+1}^{T} \gamma^{i-t-1} r_i$, where T is the terminal time-step in an episode. Q-function approximator, used by MCTS and STS in GRF experiments (see Section A.5 for details), is trained via the MSE loss and tree action-values targets, similar to the one used in Hamrick et al. [7], Miłoś et al. [10].

Policy, π_ϕ, is trained using the cross-entropy loss. As targets, we use one-hot encoded actions chosen in the environment for Random Shooting and the empirical distribution of actions chosen in the root during the planning for Bandit Shooting, MCTS, and STS.

The total loss is a weighted sum of the value function (or the Q-function) loss, the policy loss (weighted by $1\mathrm{e}{-2}$ in Random Shooting and Bandit Shooting, and $1\mathrm{e}{-3}$ in MCTS and STS), and a regularizing, l_2 term (weighted by $1\mathrm{e}{-6}$).

A pre-trained PPO policy in Shooting methods was obtained using a script included in the Google Research Football repository (see Kurach et al. [9]) and the OpenAI Baselines (Dhariwal et al. [3]) PPO2 implementation.

A.2 Hyper-parameters

	Sokoban			Google Research Football		
Parameter	Shooting	MCTS	STS	Shooting	MCTS	STS
Number of passes C	48	50	10	30	300	30
Planning horizon H	5	1	5	10	1	10
Discounting γ	0.99	0.99	0.99	0.95 / 0.99[1]	0.99	0.99
Exploration weight c_{puct}	10.0[2]	-	-	1.0 / 2.5[3]	1.0	1.0
Policy π_ϕ temperature[4]	2.0	-	-	2.0	1.0	1.0
Action sampling temp. τ	-	-	-	0.3[5]	0.3	0.3
Dirichlet parameter α	-	-	-	0.03[5]	0.3	0.3
Noise weight c_{noise}	-	-	-	0.1[5]	0.1	0.1 / 0.3[6]
Depth limit `depth limit`[7]	-	-	-	-	30	30
VF zero-initialization[8]	no	no	no	no	yes	yes
Optimizer	RMS	RMS	RMS	RMS	Adam	Adam
Learning rate	2.5e−4	2.5e−4	2.5e−4	1.0e−4	1.0e−3	1.0e−3
Batch size	32	32	32	64	64	64

[1] All $\gamma = 0.99$ except for Shooting experiments with a uniform and a pre-trained PPO policy, where $\gamma = 0.95$.
[2] Applies only to Bandit Shooting.
[3] $c_{puct} = 1.0$ for Bandit Shooting with a uniform and a pre-trained PPO policy and $c_{puct} = 2.5$ for Bandit Shooting with a trained policy.
[4] Softmax temperature. MCTS and STS in Sokoban does not use policy, see Section A.5 for details.
[5] Applies only to Bandit Shooting with additional exploration mechanisms, see Section A.4.
[6] $c_{noise} = 1.0$ for STS Conv. and $c_{noise} = 0.3$ for STS MLP.
[7] The maximum number of nodes visited in a single planning pass, see Section A.5.
[8] If the last layer of a value function neural network was initialized to 0, see Section A.8.2.

Table 3: Default values of hyper-parameters used in our experiments.

Table 3 presents hyper-parameters used in our experiments. These were based on hyper-parameters previously proposed in the literature, e.g., Miłoś et al. [10], and a certain amount of tuning.

A.3 Network architectures

In GRF experiments we use two different state representations: 'simple115' and 'extended' (see Section A.8). In the former case, we use an MLP architecture with two hidden layers of 64 neurons, while in the latter case, we use 4 convolutional layers with 16, 3x3, filters, zero-padding and stride 2, followed by a dense layer of 64 neurons. In both cases, two heads, corresponding to a value function (or Q-function for MCTS and STS) and policy, follow.

In Sokoban experiments, we use 5 convolutional layers of 64, 3x3, filters with zero-padding and stride 1, followed by a dense layer of 128 neurons and heads corresponding to a value function and policy (policy is used only for Shooting methods).

In all the cases, we use the ReLU non-linearity. We use the standard Keras initialization schemes, except for MCTS and STS in GRF experiments, see Section A.8.2.

A.4 Random Shooting

In this section we present two instantiations of Algorithm 1, which use Monte Carlo rollouts to evaluate state-actions pairs: Random Shooting and Bandit Shooting, see Algorithm 4 and Algorithm 5, respectively.

Algorithm 4 Random Shooting Planner

```
function SELECT(state)
    $s \leftarrow$ state
    $a \sim \pi_\phi(s, \cdot)$
    $s', r \leftarrow model.\text{STEP}(s, a)$
    return $(s, a, r),\ s'$
function EXPAND(leaf)
    $s_0 \leftarrow$ leaf
    rollout $\leftarrow (s_k, a_k, r_{k+1})_{k=0}^{H-1}$
     where $s_{k+1}, r_{k+1} \leftarrow model.\text{STEP}(s_k, a_k)$
     and $a_k \sim \pi_\phi(s_k, \cdot)$
    return rollout, $s_H$
function UPDATE(path, rollout, leaf)
    $\hat{G} \leftarrow \sum_{k=1}^{H} \gamma^{k-1} r_k + \gamma^H \mathbf{V}_\theta(\text{leaf})$
     where $r_k \in$ rollout
    $s, a, r \leftarrow$ path
    quality $\leftarrow r + \gamma * \hat{G}$
    $W(s, a) \leftarrow W(s, a) +$ quality
    $N(s, a) \leftarrow N(s, a) + 1$
    $Q(s, a) \leftarrow \frac{W(s,a)}{N(s,a)}$
function CHOOSE_ACTION($s$)
    return $\arg\max_a Q(s, a)$
```

The simplest version of Algorithm 4, the so-called flat Monte Carlo [5, 13], does not use a policy π_ϕ (instead rollouts are uniformly sampled) nor a value function $\mathbf{V}_\theta$ (just truncated sum of rewards $\hat{G} = \sum_{k=1}^{H} \gamma^{k-1} r_k$). Bandit Shooting, presented in Algorithm 5, is a Multi-armed Bandits variant of Random Shooting and uses PUCT [15] rule to improve exploration and thus achieve more reliable evaluations of actions.

Algorithm 5 Bandit Shooting Planner, additionally requires exploration weight c_{puct}

```
function SELECT(state)
    $s \leftarrow$ state
    $U(s, a) \leftarrow \sqrt{\sum_{a'} N(s, a')}/(1 + N(s, a))$
    $a \leftarrow \arg\max_a (Q(s, a) + c_{puct} \pi_\phi(s, a) U(s, a))$
    $s', r \leftarrow model.\text{STEP}(s, a)$
    return $(s, a, r),\ s'$
function EXPAND(leaf)
    The same as in Algorithm 4.
function UPDATE(path, rollout)
    The same as in Algorithm 4.
function CHOOSE_ACTION($s$)
    return $\arg\max_a N(s, a)$
```

Algorithm 6 Bandit Shooting Planner with additional exploration mechanisms, requires exploration weight c_{puct}, action sampling temperature τ, noise weight c_{noise} and Dirichlet distribution parameter α

```
function SELECT(state)
    $s \leftarrow$ state
    $P(s,a) \leftarrow (1 - c_{noise})\pi_\phi(s,a) + c_{noise}D$
    $U(s,a) \leftarrow \sqrt{\sum_{a'} N(s,a')}/(1 + N(s,a))$
    $a \leftarrow \arg\max_a(Q(s,a) + c_{puct}P(s,a)U(s,a))$
    $s', r \leftarrow model.\text{STEP}(s,a)$
    return $(s,a,r),\ s'$
function EXPAND(leaf)
    The same as in Algorithm 4.
function UPDATE(path, rollout)
    The same as in Algorithm 4.
function CHOOSE_ACTION($s$)
    $a \sim \text{softmax}\big(\frac{1}{\tau}\log N(s,\cdot)\big)$
    return $a$
```

Bandit Shooting with additional exploration mechanisms Algorithm 6 describes Bandit Shooting with additional exploration mechanisms: mixing the policy with Dirichlet noise (as in [15]) and action sampling with temperature τ in CHOOSE_ACTION(s). The noise variable D is sampled from the Dirchlet distribution $Dir(\alpha)$ each time when PLANNER is called (see also Algorithm 2).

A.5 MCTS

Monte Carlo Tree Search (MCTS) is a family of methods, that iteratively and explicitly build a search tree, see Browne et al. [2]. It follows the schema of Algorithm 1. `SELECT` traverses down the tree, according to an in-tree policy, until a leaf is encountered. `EXPAND` grows the tree by adding the leaf's children. The values of these new nodes are estimated, usually with the help of a rollout policy in a similar vein as Random Shooting Planner, or via the value network V_θ (see Silver et al. [14]). In this work, we refer to the latter version, using value networks, as MCTS. Finally, `UPDATE` backpropagates these values from the leaf up the tree. After planning, `CHOOSE_ACTION` chooses an action to take according to the number of visits in each child of the root node. This is consistent with AlphaZero [15]. A basic variant of MCTS is presented in Algorithm 7. More details are provided in Appendix A.5.

Algorithm 7 MCTS, additionally uses tree structure $tree$.

```
function SELECT(state)
    $s \leftarrow$ state
    path $\leftarrow$ []
    while $s$ belongs to $tree$ do
        $a \leftarrow$ CHOOSE_ACTION($s$)
        $s', r \leftarrow tree[s][a]$
        path.APPEND($(s,a,r)$)
        $s \leftarrow s'$
    return path, $s$
function EXPAND(leaf)
    for $a \in \mathcal{A}$ do
        $s', r \leftarrow model.\text{STEP}(\texttt{leaf}, a)$
        $tree[\texttt{leaf}][a] \leftarrow (s', r)$
        $W(\texttt{leaf}, a) \leftarrow r + \gamma * \mathbf{V}_\theta(s')$
        $N(\texttt{leaf}, a) \leftarrow 1$
        $Q(\texttt{leaf}, a) \leftarrow W(\texttt{leaf}, a)$
    return [], leaf
function UPDATE(path, rollout, leaf)
    quality $\leftarrow \mathbf{V}_\theta$(leaf)
    for $s, a, r \leftarrow$ reversed(path) do
        quality $\leftarrow r + \gamma *$ quality
        $W(s,a) \leftarrow W(s,a) +$ quality
        $N(s,a) \leftarrow N(s,a) + 1$
        $Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)}$
function CHOOSE_ACTION($s$)
    return $\arg\max_a Q(s,a)$
```

In our experiments, we used various implementations of MCTS. The reasons were two-fold. First, some implementation details fit better Sokoban and some GRF. Second, we wanted to check in various cases that the multi-step expansion is beneficial, see Section A.6.

In Sokoban experiments, we used the MCTS implementation similar to the one in Miłoś et al. [10], containing a loop avoidance mechanism and transposition tables. The loop avoidance mechanism alters `SELECT` and `CHOOSE_ACTION` (see Algorithm 7) so that the selected `path` does not contain repetitions of states. The transposition tables are a rather standard technique, which proposes to accumulate search statistics (i.e., W, N, Q) for states of the environment (rather than for the nodes of the search tree, as it happens in the standard case).

In GRF, we used our custom implementation of MCTS based on the one in Silver et al. [14]. It uses leaf evaluation with Q-function and policy networks. The Q-function is used to evaluate all children of a given node at once (instead of separately invoking value function $\mathbf{V}_\theta$ in `UPDATE`). The policy network is considered to be 'prior' for choosing actions, similarly as in `SELECT` in Algorithm 6. Dirichlet noise, parameterized by α and c_{noise}, is mixed with the prior in the root and action sampling with temperature τ is used to choose action on the real environment, similarly as in Bandit Shooting with additional exploration mechanisms in Section A.4. Additionally, we put a limit, `depth limit`, on the maximum number of nodes visited in a single STS pass.

A.6 STS

We tested STS with two MCTS setups described in Section A.5. In both the cases we observed substantial experimental improvements as reported in Section A.7 and Section A.8. This alone, in our view, provides enough evidence that the *multi-step expansion* is a useful method.

Apart from this, STS offers practical computational benefits, which are analyzed below.

A.6.1 Computational benefits of STS

We distinguish three types of computational costs in MCTS (see Algorithm 7):

1. Traversing down the search tree (performed in SELECT and EXPAND).
2. Backpropagation of values and counts update (handled by UPDATE).
3. Evaluation of heuristics (value network $\mathbf{V}_\theta$, or Q-function and policy as described in Section A.5)

In large GRF experiments, we found that it was the first cost that dominated the remaining two. The reason is that the cost of building a search tree is quadratic to its depth. The use of *multi-step expansion* significantly reduces this cost as several nodes are added during single tree traversal. In our case, these benefits allowed for much smoother experimenting with GRF and are, arguably, a step towards developing more efficient planners. We expect this might be practically useful (i.e., costs 1 and 2 are dominant) when the search size is large, or the heuristic evaluation is relatively cheap compared to the environment step. This is the case in some of our GRF experiments. The GRF simulator is rather complex and slower than small MLP networks.

The following simple lemma offers some theoretical analysis.

Lemma A.6.1. *Assume that STS and MCTS build the same tree $\mathcal{T}$, starting from the root state s_0. Denote the number of nodes in $\mathcal{T}$ as C and the number of nodes to be added at a single multi-step expansion of STS as H. Then the number of steps in $\mathcal{T}$ performed by STS will be lower compared to MCTS by a factor in $[\frac{h-1}{2}, h]$.*

Proof. Lets consider h consecutive nodes $s_1, \dots, s_h$ in the search tree added in a single EXPAND step during STS search. In STS, the number of steps, C_{STS}, in the tree during SELECT and EXPAND is equal to $h+d$, where d is distance between s_0 and s_1 in $\mathcal{T}$. To add the same set of nodes during MCTS search, one need h separate calls to SELECT and EXPAND. The total number of steps performed is $C_{MCTS} = \sum_{k=0}^{h-1} d + k + 1 = hd + h\frac{h-1}{2}$. Clearly,

$$\frac{h-1}{2} C_{STS} \leq C_{MCTS} \leq h C_{STS}.$$

Similar calculation hold for the costs of backpropagation. □

A.7 Sokoban experiments

Sokoban is a well-known combinatorial puzzle, where the agent's goal is to push all boxes (marked as yellow, crossed squares) to the designed spots (marked as squares with a red dot in the middle), see Figure 2. Additionally, to the navigational challenge, Sokoban's difficulty is attributed to the irreversibility of certain actions. A typical example is pushing a box into a corner, though there are

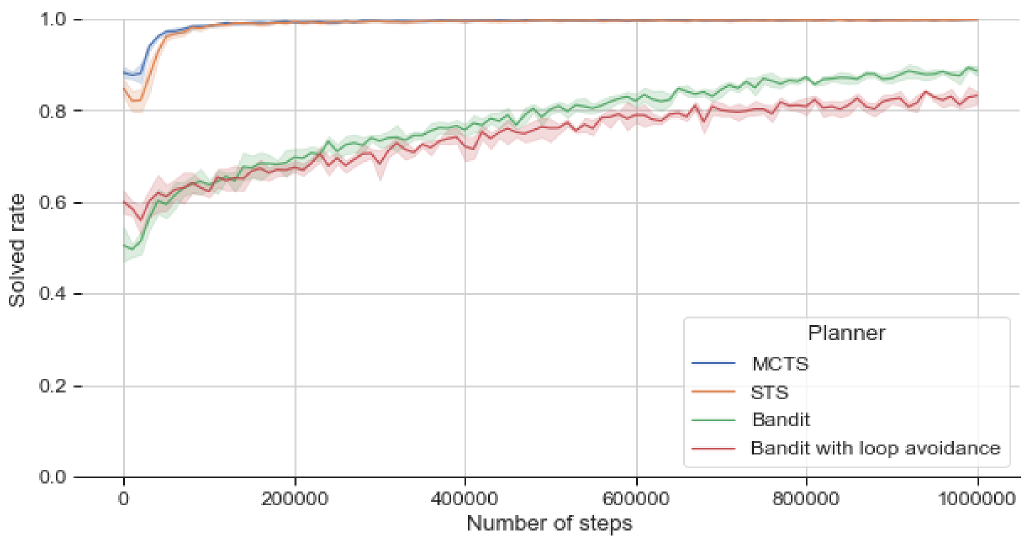


Figure 3: Sokoban on simpler boards: training curves for MCTS, STS and Bandit Shooting with and without loop avoidance. Mean over 5 seeds with shaded regions representing 95% confidence intervals.

multiple less apparent cases. The environment's complexity is formalized by the fact that, deciding whether a level of Sokoban is solvable or not, is PSPACE-complete, see e.g. Dor and Zwick [4]. Due to these challenges, the game is often used to test reinforcement learning and planning methods.

Sokoban is an environment known for its combinatorial complexity. The agent's goal is to push all boxes (marked as yellow, crossed squares) to the designed spots (marked as squares with a red dot in the middle), see Figure 2. Apart from the navigational challenge, the difficulty of this game is greatly increased by the fact that some actions are irreversible. A canonical example of such an action is pushing a box into a corner, though there are multiple less obvious cases. Formally, this difficulty manifests itself in the fact that deciding whether a level of Sokoban is solvable or not, is NP-hard, see e.g. Dor and Zwick [4]. Due to these challenges the game has been considered as a testbed for reinforcement learning and planning methods.

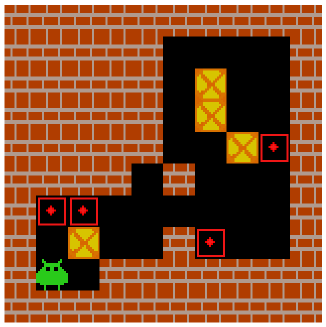


Figure 2: Example $(10, 10)$ Sokoban board with 4 boxes. Boxes (yellow) are to be pushed by agent (green) to designed spots (red). The optimal solution in this level has 37 steps.

For a description of Sokoban see Section 4.1. In our experiments, we used inputs of dimension $(x, x, 7)$, where (x, x) is the size of the board ($(10, 10)$ in most cases) and 7 is one-hot encoding of the state of a given cell (enumerated as follows: wall, empty, target, box_target, box, player, player_target). In most experiments, we used 4 boxes and the limit of 200 steps. The agent is rewarded with 1 by putting a box into a designated spot and additionally with 10 when all the boxes are in place[2]. The action space consists of four movement directions (up, down, right, left).

A.7.1 Evaluation experiments

In Table 4 we show full details of the evaluation experiment (which complements Table 1). Recall that in this experiment, we evaluated the planning capabilities of STS in isolation from training. To this end, we used a pre-trained value function and varied the number of passes C and the depth of multi-step expansion H, such that $H \cdot C$ remains constant. In Table 4, we present quantities (N_p, N_t, N_g), which measure planning costs for finding a solution (the average number of passes,

[2]Our Sokoban code is fully compatible with Racanière et al. [11].

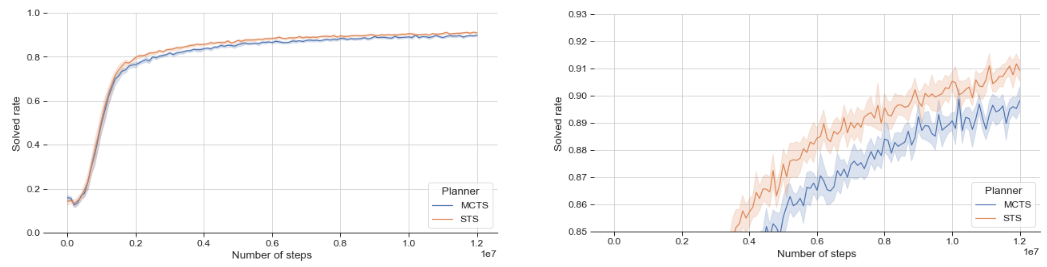


Figure 4: Learning curve for Sokoban domain. Left figure shows full results, right one inspects the same data for limited interval of values on the y axis. The results are averaged over 10 runs, shaded areas shows 95% confidence intervals. The x axis is the number of collected samples.

Scenario	C	H	S. rate	N_p	N_t	N_g
avoid loops	256	1	95.2%	1224	1224	716
	128	2	95.9%	569	1137	728
	64	4	96.5%	299	1194	830
	32	8	95.9%	173	1385	1040
	16	16	95.7%	114	1822	1333
	8	32	93.4%	79	2527	1528
	4	64	89%	62	3960	1491
	2	128	80%	52.7	6754	1207
no avoid loops	256	1	84.5%	1497	1497	376
	128	2	86.3%	724	1448	332
	64	4	87.8%	385	1541	370
	32	8	88.4%	185	1483	409
	16	16	89.5%	110	1754	539
	8	32	89.9%	84	2690	882
	4	64	85.2%	68	4463	1300
	2	128	65.3%	36	4589	967

Table 4: Evaluation of various STS settings on Sokoban

tree nodes and game states observed, respectively, until the solution is found). We run experiments with and without the loop avoidance mechanism (see Section A.5). We observe that there is a sweet spot for the choice of H. It is evident for the 'no avoid loop' case, $C = 32, H = 8$. For this choice, the number of tree nodes, N_t, which is the most important metric, is the smallest. Interestingly, we observe a significant increase in the solved rate. This may be explained by the fact that the number of distinct visited game states, N_g, grows. This suggests that STS explores more aggressively and efficiently. For bigger H, we observe a further increase of the solved rate until some point, though at the cost of much bigger N_t.

In experiments with the avoid loop mechanism, there is a similar effect for $C = 64, H = 4$, though more subtle, probably because results are already quite strong. Moreover, we observe a more significant drop in performance as H increases (when planning resembles more shooting methods).

The values presented in Table 4 are averages over more than 5000 boards.

A.7.2 MCTS and Shooting on simpler boards

We found the Bandit Shooting method underperforming on Sokoban. As a sanity test, we tested a simpler setting with smaller boards of size $(6, 6)$ and two boxes. Learning curves are presented in Figure 3. MCTS and STS experiments quickly learn to solve over 99% of boards. Bandit Shooting experiment showed stable but much slower progress. We also evaluated the version of Bandit Shooting, with additional loop avoidance, see Section A.5. This mechanism was beneficial for MCTS and STS but failed to bring improvements for the shooting algorithms.

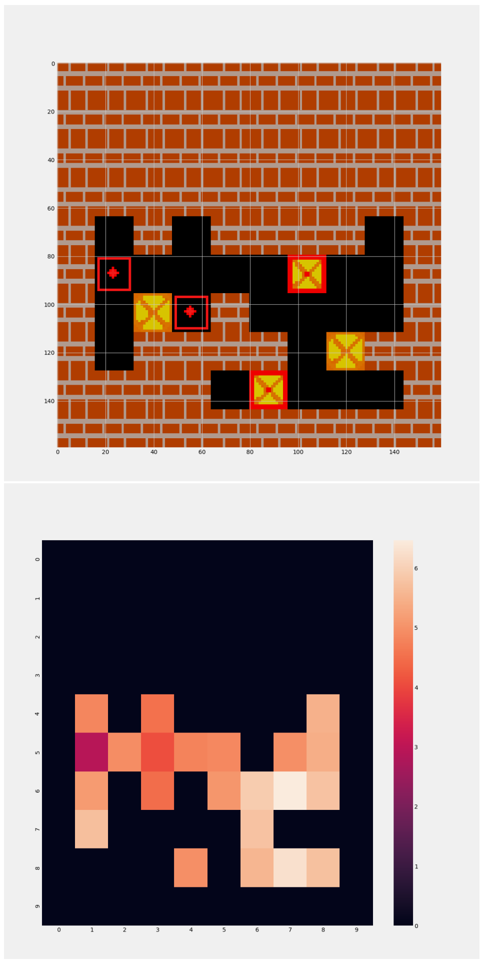


Figure 5: Sokoban value function heatmap, brighter means higher value estimate.

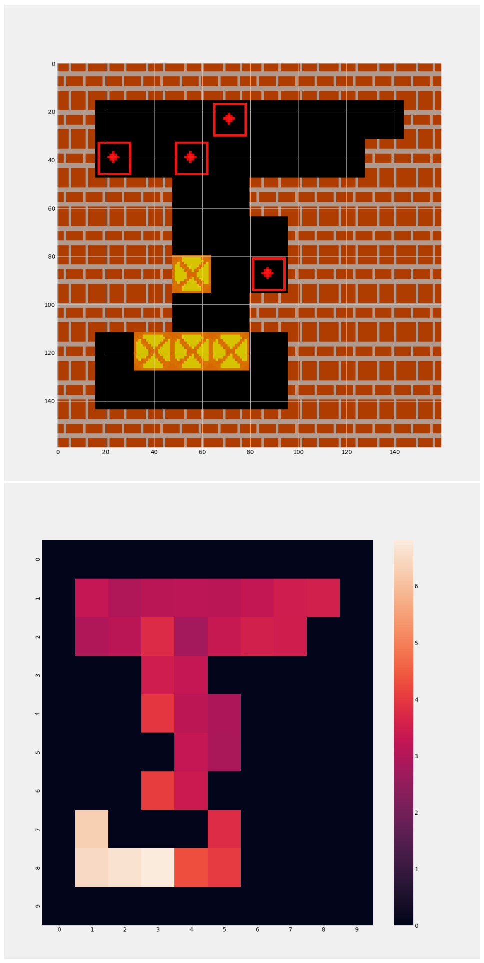


Figure 6: Sokoban value function heatmap, brighter means higher value estimate.

A.7.3 Correcting biased value functions estimates with deep search

To generate value function heatmaps we evaluated the pre-trained MCTS value function for each possible agent position in a room. Figures 5 and 6 present two chosen rooms with their corresponding VF heatmaps. Specifically, the room in Figure 6 was solved by the STS with 10 passes and 5 steps of multi-step expansion and wasn't solved by the MCTS with 50 passes, both without the avoid loops mechanism. We include movies of both agents in this room in the code repository: `https://github.com/shoot-tree-search/sts/tree/master/movies`.

Because the value function is biased, it makes MCTS stuck in states with overestimated value. See Figure 6, in this room MCTS gets stuck in the bottom-left region. However, with a deeper tree, STS can get unstuck quicker and still find a solution. Remember that the search statistics (i.e., W, N, Q) are accumulated for states of the environment (see Appendix A.5). As this overestimated region gets searched deeper the bias in the value function gets discounted more and the agent figures out there are no rewards in reality. At some point, other actions will have a higher value and the agent has a chance to get unstuck and explore other parts of the room. That being said, it should be noted that this "potential well" will still attract the agent, make its planning paths distracted, even when it gets unstuck. STS is less vulnerable to this effect and is able to solve this room despite high bias in the value function.

A.8 Google Research Football experiments

For a description of Google Research Football see Appendix A.8.

Google Research Football (GRF) is an environment recently introduced in Kurach et al. [8]. It is an advanced, physics-based simulator of the game of football. A part of GRF is the Football Academy consisting of 11 scenarios highlighting various tactical difficulties, see Kurach et al. [8, Table 10] for description. A GRF Academy episode is considered finished after 100 steps or when the goal is scored by the agent.

Reported results correspond to solved rates over 20 episodes in case of Shooting methods with an uniform and a pre-trained policy and around 30 episodes in case of all other methods. Results for MCTS, STS, and Shooting methods with the trained policy are medians of at least three training runs. During evaluations we disabled Dirichlet noise and action sampling (in Bandit Shooting Expl., MCTS and STS).

Google Research Football offers two major mode of observations: 'simple115' and 'extracted' (also called the super mini-map).

The simple115 state representation is consists of coordinates of players, players' movement directions, the ball position, a ball movement direction, a one-hot encoding of ball ownership, a one-hot encoding of which player is active. This totals in a vector of length 115.

The extracted state representation consists 4 stacked layers of size $(72, 96)$. Layers contain one-hot encoding of spatial positions of game entities. These are (on the subsequent layers): players on the left team, players on the right team, the ball, and the active player.

We note that even though the extracted representation contains 'less information' than simple115, it has been reported in [8] to generate better results.

In our experiments, we use the so-called checkpoint rewards, which provide an additional signal for approaching the goal area. Details can be found in [8], where they were introduced and used in large-scale experiments.

The action space in GRF consists of 19 actions representing high-level football behaviors (e.g. "Short Pass"), see Kurach et al. [8, Table 1].

Figure 7 shows selected training curves on Google Research Football, the best from each family of our methods: Shooting, MCTS and STS. Figure 8 shows all training curves for our methods on Google Research Football. On the y-axis is the solved rate calculated as described above in Section A.8. On the x-axis is the number of real steps in the environment (planning steps in the simulator are not added). Curves are mean over 3 training runs with different seeds and shaded regions represent 95% confidence intervals (exceptionally for Bandit Shooting we report just one run). Moreover, to smooth the curves, data points are averaged in the windows of 10000 steps.

A.8.1 Shooting methods

Table 5 presents our methods performance in all Google Research Football academies.

Tuning c_{puct} turned out to be the most important one to make Bandit Shooting work, see Algorithm 5. In a nutshell, it needs to be adjusted to scale of rewards (value function) in a given environment. In our experiments we found $c_{puct} = 2.5$ to work best.

Using additional Dirichlet noise, $c_{noise} > 0$, and action sampling on the real environment, $\tau > 0$ (see Algorithm 6) resulted in inferior results with an exception of the "Counterattack hard" scenario.

A.8.2 MCTS and STS experiments

Apart from *multi-step expansion* we introduced another simple novel method, which might be of interest to the general public. Namely, before starting training, we set the weights of the last layer of the Q-value neural network to 0 (see Section A.3 for a detailed description of architectures). We observed that this significantly improved the training stability due to better exploration (and avoiding suboptimal strategies at the early stages of training). See 'No zero initialization' on Figure 10.

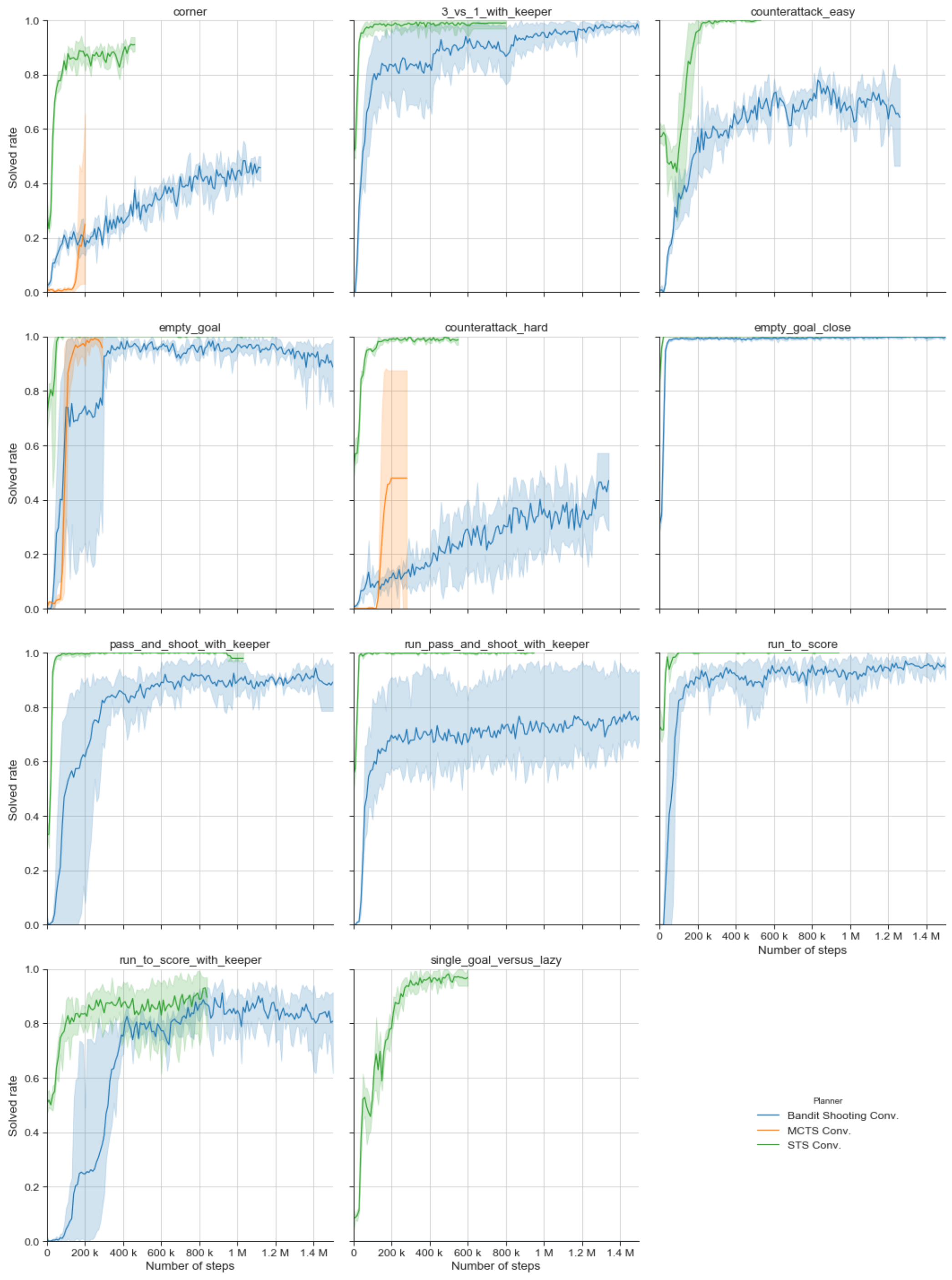


Figure 7: Google Research Football training curves for three best methods on GRF. Mean over 3 seeds with shaded regions representing 95% confidence intervals.

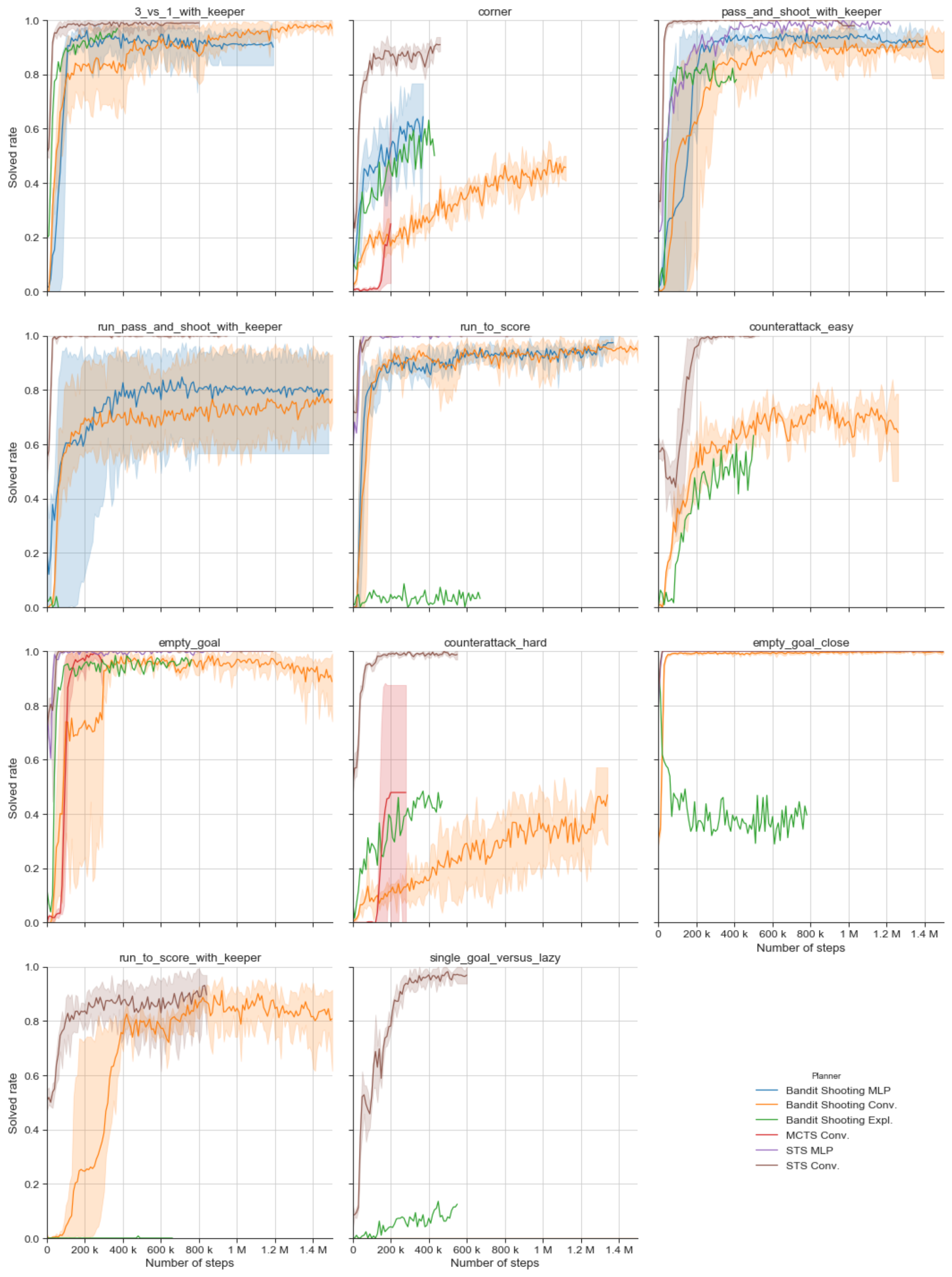


Figure 8: Google Research Football training curves for all methods on GRF. Mean over 3 seeds with shaded regions representing 95% confidence intervals.

Method		3 vs. 1 with keeper	Corner	Counterattack easy	Counterattack hard	Empty goal	Empty goal close	Pass and shoot with keeper	Run pass and shoot with keeper	Run to score	Run to score with keeper	Single goal versus lazy
	PPO	0.90	0.10	0.70	0.65	0.90	**1.00**	0.65	0.90	0.90	**1.00**	0.90
Random Shooting	flat	0.10	0.00	0.05	0.10	0.00	0.95	0.05	0.10	0.00	0.00	0.00
	PPO	0.45	0.10	0.10	0.30	**1.00**	**1.00**	0.25	0.80	0.80	0.20	0.30
	MLP	0.90	**0.87**	0.80	0.73	0.93	**1.00**	0.87	0.70	0.90	0.37	0.67
Bandit Shooting	flat	0.20	0.10	0.00	0.00	0.05	0.35	0.05	0.05	0.05	0.00	0.00
	PPO	**1.00**	0.05	0.95	0.80	**1.00**	**1.00**	0.55	**1.00**	0.85	0.45	0.60
	MLP	0.87	0.47	0.73	0.60	**1.00**	**1.00**	0.90	0.80	0.93	**1.00**	0.60
	Conv.	0.97	0.41	0.81	0.44	0.97	**1.00**	0.94	0.69	**1.00**	0.91	0.00
	Expl.	**1.00**	0.53	0.50	0.66	**1.00**	0.00	0.81	0.34	0.00	0.00	0.09
MCTS Conv.		0.81	0.50	0.31	0.31	0.99	**1.00**	0.45	0.89	0.70	0.00	0.00
STS	MLP	**1.00**	0.78	**1.00**	0.97	**1.00**	**1.00**	0.94	0.97	**1.00**	0.94	0.94
	Conv.	**1.00**	0.81	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.97	**0.97**

Table 5: Summary of methods performance on GRF. Entries correspond to rounded solved rates over at least 20 episodes per environment. Results for Shooting methods with the trained policy, MCTS and STS are reported as median of at least three training runs. PPO results come from Kurach et al. [8].

A.8.3 Ablations

The ablations were performed on three environments from GRF Academy: *corner*, *counterattack hard* and *empty goal*, see Figure 10. The first two environments are difficult, while the last one is easy. The following parameters or settings were subject to analysis (they correspond to the labels in Figure 10):

- `prior noise weight`: a weight in the mixture of Dirichlet noise and the prior.
- `depth limit`: the maximum number of nodes visited in a single STS pass.
- `sampling temperature`: temperature for sampling the actions on the real environment.
- `MCTS n_passes 300`: this corresponds the standard MCTS setting with $H = 1$ (MCTS) and $C = 300$
- `Value network n_passes`: value network is used instead of Q-function. Note that `n_passes` $= 2$ matches roughly the Q-function version in terms of visited states (recall, see Section 7, that Q-function evaluates all children at once and that number of actions in GRF is 19).
- `No policy`: instead of a learned policy network, a uniform policy is used.
- `No zero initialization`: the last layer of the value function neural network was not initialized to 0 (see description at the beginning of Section A.8.2).

The default setup (denoted as `Prior noise weight 0.1`) is always positioned at the top in Figure 10. It uses parameters described in Table 3 in the Google Research Football STS column.

A.9 Multi-step expansion analysis on toy problems

First, consider an MDP presented at the top of Figure 9. It showcases the situation when the errors are systematic: in the vicinity of the starting state s_0, the estimates of the value function are biased (for simplicity set to 0 and shown as white vertices), while the values in the area surrounding terminal states are accurate (shown as color vertices). This example is an exaggeration. However, something similar can happen in practice, when information is propagated with TD-like methods or the environment has an “easy” region, which is hard to find. Under these circumstances, STS, given large enough H, will be able to reach accurate values (color vertices) within a few passes. On the contrary, MCTS would explore the whole uncertain area (white vertices) in a breadth-first fashion.

Second, consider an MDP shown at the bottom of Figure 9. It illustrates the case when the errors are "pseudo-random". In this MDP all rewards are 0 except the marked edges, where they are $-a, a > 0$. Starting from s_0, the agent can move only to the right. The perfect value function is 0 in every state, however we assume that the current noisy value estimates equal to ϵ_i on the "tail" part of the diagram. In this example, we assume that the errors arise in interactions of many factors, thus can be modeled as i.i.d. centered random variables ϵ_i such that $\mathbb{E}|\epsilon_i| < +\infty$.

The optimal path, going over the green edge and later over the tail, is accompanied by several 'decoy' paths (marked in orange). They will not be entered unless errors on the tail have accumulated below $-a$. We denote the probability of such an event by p_H, where H is the number of steps in the multi-step expansion ($H = 1$ corresponds to MCTS). In Lemma A.9.1, we show that $p_1 > p_H$ for $H \geq 2$, and in fact $p_H \to 0$ when $H \to +\infty$.

Lemma A.9.1. *Under the above assumptions $p_1 > p_H$ and $p_H \to 0$.*

Proof. Assume that for the first $\ell \geq 2$ steps of the search tree was unfolded via the middle (green) edge and further via the tail. The state-action value estimated by the MCTS/STS is thus $q_\ell = (\epsilon_0 + \ldots + \epsilon_{\ell-2})/\ell$. Consequently,

$$p_H = \mathbb{P}(\exists_{k \in \mathbb{N}} q_{kH} < -a).$$

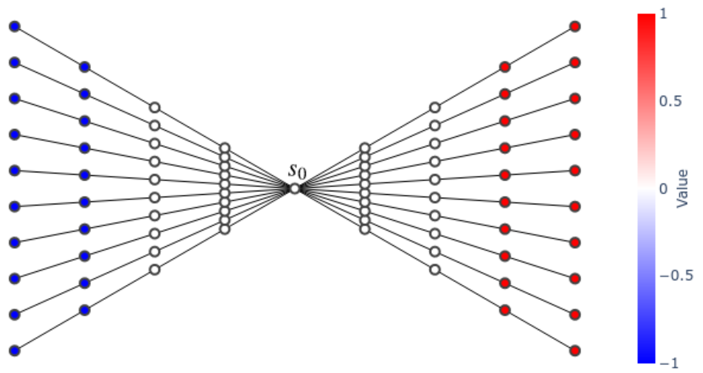
The claims follow from the fact $q_\ell \to 0$ a.s., which itself is the consequence of the strong law of large numbers. □

As the lemma serves mainly the illustrative purpose we used the i.i.d. assumption, which can be easily weakened. As a test we simulate the case $\epsilon_i \sim \mathcal{N}(0, 1)$ and $a = 0.3$. In this case $p_1 = 0.56, p_2 = 0.46, p_4 = 0.35, p_8 = 0.41, p_{16} = 0.21$. Note that p_1/p_H is as high as 3 for $H = 16$ and quite natural choice of a and ϵ_i.

A.10 Infrastructure used

We ran our experiments on clusters with servers typically equipped with 24 or 28 CPU cores and 64GB of memory. A typical experiment was 72 hours long (the timeout set on the clusters), which was enough for most experiments. Experiments that did not converge during this time were resumed.

During the project, we run more than 10k experiments.



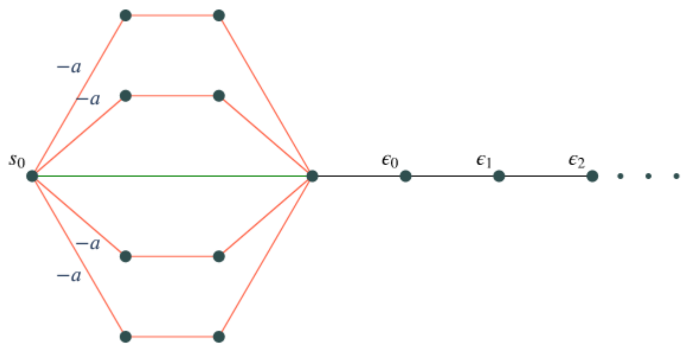


Figure 9: Visualization of the toy environments.

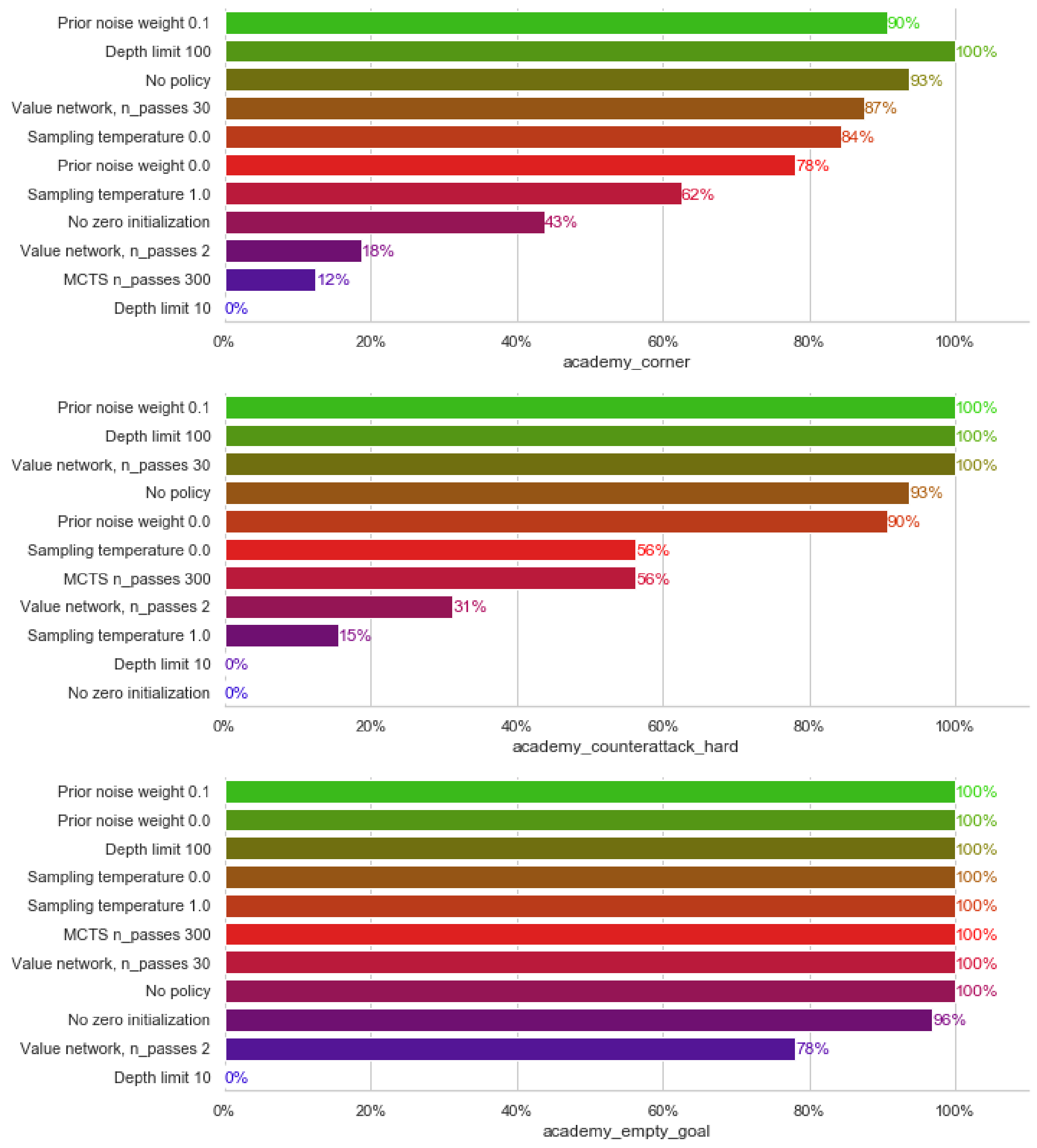


Figure 10: Ablations performed GRF Academy environments: *corner*, *counterattack hard* and *empty goal*.