Subgoal Search For Complex Reasoning Tasks

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

Tomasz Odrzygóźdź[*]
University of Warsaw
tomaszo@impan.pl

Marek Zbysiński
University of Warsaw
m.zbysinski@students.mimuw.edu.pl

Michał Zawalski
University of Warsaw
m.zawalski@uw.edu.pl

Krzysztof Olejnik
University of Warsaw
k.olejnik3@student.uw.edu.pl

Yuhuai Wu
University of Toronto,
Vector Institute
ywu@cs.toronto.edu

Łukasz Kuciński
Polish Academy of Sciences
lkucinski@impan.pl

Piotr Miłoś
Polish Academy of Sciences,
University of Oxford,
deepsense.ai
pmilos@impan.pl

Abstract

Humans excel in solving complex reasoning tasks through a mental process of moving from one idea to a related one. Inspired by this, we propose Subgoal Search (kSubS) method. Its key component is a learned subgoal generator that produces a diversity of subgoals that are both achievable and closer to the solution. Using subgoals reduces the search space and induces a high-level search graph suitable for efficient planning. In this paper, we implement kSubS using a transformer-based subgoal module coupled with the classical best-first search framework. We show that a simple approach of generating k-th step ahead subgoals is surprisingly efficient on three challenging domains: two popular puzzle games, Sokoban and the Rubik's Cube, and an inequality proving benchmark INT. kSubS achieves strong results including state-of-the-art on INT within a modest computational budget.

1 Introduction

Reasoning is often regarded as a defining property of advanced intelligence [40, 18]. When confronted with a complicated task, humans' thinking process often moves from one idea to a related idea, and the progress is made through milestones, or *subgoals*, rather than through atomic actions that are necessary to transition between subgoals [15]. During this process, thinking about one subgoal can lead to a possibly diverse set of subsequent subgoals that are conceptually reachable and make a promising step towards the problem's solution. This intuitive introspection is backed by neuroscience evidence [20], and in this work, we present an algorithm that mimics this process. Our approach couples a deep learning generative subgoal modeling with classical search algorithms to allow for successful planning with subgoals. We showcase the efficiency of our method on the following complex reasoning tasks: two popular puzzle games Sokoban and the Rubik's Cube, and an inequality theorem proving benchmark INT [55], achieving the state-of-the-art results in INT and competitive results for the remaining two.

The deep learning revolution has brought spectacular advancements in pattern recognition techniques and models. Given the hard nature of reasoning problems, these are natural candidates to provide

[*]equal contribution

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

search heuristics [4]. Indeed, such a blend can produce impressive results [44, 45, 37, 1]. These approaches seek solutions using elementary actions. Others, e.g. [30, 34, 24], utilize variational subgoals generators to deal with long-horizon visual tasks. We show that these ideas can be pushed further to provide algorithms capable of dealing with combinatorial complexity.

We present Subgoal Search (kSubS) method and give its practical implementations: MCTS-kSubS and BF-kSubS. kSubS consists of the following four components: planner, subgoal generator, a low-level policy, and a value function. The planner is used to search over the graph induced by the subgoal generator and is guided by the value function. The role of the low-level policy is to prune the search tree as well as to transition between subgoals. In this paper, we assume that the generator predicts subgoals that are k step ahead (towards the solution) from the current state, and to emphasize this we henceforth add k to the method's abbreviation. MCTS-kSubS and BF-kSubS differ in the choice of the search engine: the former uses Monte-Carlo Tree Search (MCTS), while the latter is backed by Best-First Search (BestFS). We provide two sets of implementations for the generator, the low-level policy, and the value functions. The first one uses transformer architecture [49] for each component, while the second utilizes a convolutional network for the generator and the value function, and the classical breadth-first search for the low-level policy. This lets us showcase the versatility and effectiveness of the approach.

The subgoal generator lies at the very heart of Subgoal Search, being an implementation of reasoning with high-level ideas. To be useful in a broad spectrum of contexts, the generator should be implemented as a learnable (generative) model. As a result, it is expected to be imperfect and (sometimes) generate incorrect predictions, which may turn the search procedure invalid. Can we thus make planning with learned subgoals work? In this paper, we answer this question affirmatively: we show that the autoregressive framework of transformer-based neural network architecture [49] leads to superior results in challenging domains.

We train the transformer with the objective to predict the k-th step ahead. The main advantages of this subgoal objective are simplicity and empirical efficiency. We used expert data to generate labels for supervised training. When offline datasets are available, which is the case for the environments considered in this paper[2], such an approach allows for stable and efficient optimization with high-quality gradients. Consequently, this method is often taken when dealing with complex domains (see e.g. [42, 52]) or when only an offline expert is available[3]. Furthermore, we found evidence of out-of-distribution generalization.

Finally, we formulate the following hypothesis aiming to shed some light on why kSubS is successful: we speculate that subgoal generation may alleviate errors in the value function estimation. Planning methods based on learning, including kSubS, typically use imperfect value function-based information to guide the search. While traditional low-level search methods are susceptible to local noise, subgoal generation allows for evaluations of the value functions at temporally distant subgoals, which improves the signal-to-noise ratio and allows a "leap over" the noise.

To sum up, our contributions are:

1. We propose Subgoal Search method with two implementations: MCTS-kSubS, BF-kSubS. We demonstrate that our approach requires a relatively little search or, equivalently, is able to handle bigger problems. We also observe evidence of out-of-distribution generalization.
2. We provide evidence that a transformer-based autoregressive model learned with a simple supervised objective to predict states k-th step ahead is an effective tool to generate valid and diverse subgoals.
3. We show in our experiments that using subgoal planning help to might mitigate the negative influence of value function errors on planning.

We provide the code of our method and experiment settings at `https://github.com/subgoal-search/subgoal-search`, and a dedicated website `https://sites.google.com/view/subgoal-search`.

[2]The dataset for INT or Sokoban can be easily generated or are publicly available. For the Rubik's Cube, we use random data or simple heuristic (random data are often sufficient for robotic tasks and navigation.)

[3]For example, the INT engine can easily generate multiple proves of random statements, but *cannot* prove a given theorem.

2 Related work

In classical AI, reasoning is often achieved by *search* ([40]). Search rarely can be exhaustive, and a large body of algorithms and heuristics has been developed over the years, [40, Section 3.5]. It is hypothesized that progress can be achieved by combining search with learning [4]. Among notable successful examples of this approach are Alpha Zero [43], or solving Rubik's cube using a learned heuristic function to guide the A* algorithm (see [1]).

An eminent early example of using goal-directed reasoning is the PARADISE algorithm ([53]). In deep learning literature, [25] was perhaps the first work implementing subgoal planning. This was followed by a large body of work on planning with subgoals in the latent space for visual tasks ([24, 31, 34, 30, 21, 9, 35]) or landmark-based navigation methods ([41, 27, 14, 46, 56, 23]).

The tasks considered in the aforementioned studies are often quite forgiving when it comes to small errors in the subgoal generation. This can be contrasted with complex reasoning domains, in which even a small variation of a state may drastically change its meaning or render it invalid. Thus, neural networks may struggle to generate semantically valid states ([4, Section 6.1]).

Assuming that this problem was solved, a generated subgoal still remains to be assessed. The exact evaluation may, in general, require exhaustive search or access to an oracle (in which case the original problem is essentially solved). Consequently, it is unlikely that a simple planner (e.g., one unrolling independent sequences of subgoals [9]) will either produce an informative outcome, could be easily improved using only local changes via gradient descent [31], or cross-entropy method (CEM) [30, 34, 35]. Existing approaches, which are based on more powerful subgoal search methods, have their limitations, on the other hand. [13] is perhaps the closest to our method and uses MCTS to search the subgoal-induced graph. However, it uses a predefined (not learned) predicate function as a subgoal generator, limiting applicability to the problems with available high-quality heuristics. Learned subgoals are used in [32], a method of hierarchical planning. That said, the subgoal space needs to be relatively small for this method to work (or crafted manually to reduce cardinality). To the best of our knowledge, our algorithm is the first domain agnostic hierarchical planner for combinatorially complex domains.

More generally, concepts related to goals and subgoals percolated to reinforcement learning early on, leading, among others, to prominent ideas like hindsight [22], hierarchical learning [48, 7] or the Horde architecture [47]. Recently, with the advent of *deep* reinforcement learning, these ideas have been resurfacing and scaled up to deliver their initial promises. For example, [51] implements ideas of [7] and a very successful hindsight technique [2] is already considered to be a core RL method. Further, a recent paper [36] utilizes a maximum entropy objective to select achievable goals in hindsight training and [50] uses meta-learned subgoals to discover options in multi-task RL settings. Apart from the aforementioned hierarchical planning, the idea to use neural networks for subgoal generation was used to improve model-free reinforcement learning agents [11, 6] and imitation learning algorithms [33].

3 Method

Our method, Subgoal Search (kSubS), is designed for problems, which can be formulated as a search over a graph with a known transition model. Formally, let $G = (\mathcal{S}, \mathcal{E})$ be a directed graph and $\tilde{\mathcal{S}} \subset \mathcal{S}$ be the set of success states. We assume that, during the solving process, the algorithm can, for a given node g, determine the edges starting at g and check if $g \in \tilde{S}$.

Subgoal Search consists of four components: planner, subgoal generator, low-level policy, and a value function. The planner, coupled with a value function, is used to search over the graph induced by the subgoal generator. Namely, for each selected subgoal, the generator allows for sampling the candidates for the next subgoals. Only these reachable by the low-level policy are used. The procedure continues until the solution is found or the computational budget is exhausted. Our method searches over a graph $\tilde{G} = (\mathcal{S}, \mathcal{E}_s)$, with the edges $\mathcal{E}_s$ given by the subgoal generator. Provided a reasonable generator, paths to $\tilde{\mathcal{S}}$ are shorter in $\tilde{G}$ than in G and thus easier to find.

In this paper we provide BestFS- and MCTS- backed implementations kSubS. Algorithm 1 presents BF-kSubS; see Algorithm 4 in Appendix A.1 for MCTS-kSubS.

For INT and Rubik's Cube, we use transformer models (see Appendix B.1) in all components other than the planner. For Sokoban, we use convolutional networks (see Appendix B.2). While transformers could also be used in Sokoban, we show that a simplified setup already achieves strong results. This showcases that Subgoal Search is general enough to work with different design choices. In Section 4.2 we describe datasets used train these neural models.

Algorithm 1 Best-First Subgoal Search (BF-kSubS)

Require: C_1 max number of nodes
V value function network
SOLVED predicate of solution

function SOLVE($\mathtt{s}_0$)
  T.PUSH($(V(\mathtt{s}_0), \mathtt{s}_0)$) ▷ T is priority queue
  $\mathtt{paths}[\mathtt{s}_0] \leftarrow []$ ▷ paths is dict of lists
  $\mathtt{seen}$.ADD($\mathtt{s}_0$) ▷ seen is set
  while $0 <$ LEN(T) and LEN(seen) $< C_1$ **do**
    _, s ← T.EXTRACT_MAX()
    subgoals ← SUB_GENERATE(s)
      ▷ see Algorithm 3
    for $\mathtt{s}'$ **in** subgoals **do**
      if $\mathtt{s}'$ **in** seen **then** continue
      seen.ADD($\mathtt{s}'$)
      actions ← GET_PATH(s, $\mathtt{s}'$)
        ▷ see Alg 2 or Alg 9
      if actions.EMPTY() **then**
        continue
      T.PUSH($(V(\mathtt{s}'), \mathtt{s}')$)
      $\mathtt{paths}[\mathtt{s}'] \leftarrow \mathtt{paths}[\mathtt{s}] + \mathtt{actions}$
      if SOLVED($\mathtt{s}'$) **then**
        return $\mathtt{paths}[\mathtt{s}']$
  return False

Algorithm 2 Low-level conditional policy

Require: C_2 steps limit
π low-level conditional policy network
M model of the environment

function GET_PATH($\mathtt{s}_0$, subgoal)
  step ← 0
  s ← $\mathtt{s}_0$
  action_path ← []
  while step $< C_2$ **do**
    action ← π.PREDICT(s, subgoal)
    action_path.APPEND(action)
    s ← M.NEXT_STATE(s, action)
    if s = subgoal **then**
      return action_path
    step ← step + 1
  return []

Subgoal generator Formally, it is a mapping $\rho\colon \mathcal{S} \to \mathbf{P}(\mathcal{S})$, where $\mathbf{P}(\mathcal{S})$ is a family of probability distributions over the environment's state space $\mathcal{S}$. More precisely, let us define a trajectory as a sequence of state-action pairs $(s_0, a_0), (s_1, a_1), \ldots, (s_n, a_n)$, with $(s_i, a_i) \in \mathcal{S} \times \mathcal{A}$, where $\mathcal{A}$ stands for the action space and n is the trajectory's length. We will say that the generator predicts k-*step ahead* subgoals, if at any state s_ℓ it aims to predict $s_{\min(\ell+k,n)}$. We show, perhaps surprisingly, that this simple objective is an efficient way to improve planning, even for small values of k, i.e. $k \in \{2, 3, 4, 5\}$.

Operationally, the subgoal generator takes as input an element of the $\mathbf{s} \in \mathcal{S}$ and returns *subgoals*, a set of new candidate states expected to be closer to the solution and is implemented by Algorithm 3. It works as follows: first, we generate C_3 subgoal candidates with SUB_NET_GENERATE. Then, we prune this set to obtain a total probability greater than C_4.

For INT and Rubik's Cube, we represent states as sequences modeled with a transformer. Following the practice routinely used in language modeling, [49], SUB_NET_GENERATE employs beam search to generate a set of high likelihood outputs.[4] With the same goal in mind, for Sokoban, we use another method described Appendix C.1. SUB_NET_GENERATE uses the subgoal generator network, which is trained in a supervised way on the expert data, with training examples being: s_ℓ (input) and $s_{\min(\ell+k,n)}$ (label), see Appendix D for details.

Low-level conditional policy Formally, it is a mapping $\pi\colon \mathcal{S} \times \mathcal{S} \to \mathcal{A}^*$. It is used to generate a sequence of actions on how to reach a subgoal starting from a given initial state. Operationally, it may return an empty sequence if the subgoal cannot be reached within C_2 steps, see Algorithm 2. This is used as a pruning mechanism for the *subgoals* set in Algorithm 1.

[4] In language modeling, typically, only one beam search output is used. In our case, however, we utilize all of them, which turns out to be a diverse set of subgoals.

In INT and Rubik's Cube, we use Algorithm 2, which utilizes low-level policy network π. Similarly to the subgoal generator, it is trained using expert data in a supervised fashion, i.e. for a pair $(s_\ell, s_{\min(\ell+i,n)})$, with $i \leq k$, its objective is to predict a_ℓ.

When the branching factor is small, the low-level policy can be realized by a simple breadth-first search mechanism, see Algorithm 9 in Appendix, which we illustrate on Sokoban.

Algorithm 3 Subgoal generator

Require: C_3 number of subgoals
C_4 target probability
ρ subgoal generator network

function SUB_GENERATE(s)
  subgoals $\leftarrow \emptyset$
  states, probs $\leftarrow$ SUB_NET_GENERATE$(\rho, \mathtt{s}; C_3)$
    $\triangleright$ (states, probs) is sorted wrt probs
  total_p $\leftarrow 0$
  for state, p $\in$ (states, probs) **do**
    if total_p $> C_4$ **then break**
    subgoals.ADD(state)
    total_p $\leftarrow$ total_p + p
  return subgoals

Value function Formally, it is a mapping $V\colon \mathcal{S} \to \mathbb{R}$, that assigns to each state a value related to its distance to the solution, and it is used to guide the search (see Algorithm 1 and Algorithm 4). For its training, we use expert data. For each state s_ℓ the training target is negated distance to the goal: $\ell - n$, where n denotes the end step of a trajectory that s_ℓ belongs to.

Planner This is the engine that we use to search the subgoal-induced graph. In this paper, we use BestFS (Algorithm 1) and MCTS (Algorithm 4 in Appendix A.1). The former is a classic planning method, which maintains a priority queue of states waiting to be explored, and greedily (with respect to their value) selects elements from it (see, e.g., [40]). MCTS is a search method that iteratively and explicitly builds a search tree, using (and updating) the collected node statistics (see, e.g., [5]). In this paper, we use an AlphaZero-like [42] algorithm for single-player games.

We note that the subgoal generator can be combined with virtually any search algorithm and can benefit from an additional structure. For example, for domains providing a factored state representation, the width-based methods [26, 12] would likely be stronger search mechanisms.

4 Experiments

In this section, we empirically demonstrate the efficiency of MCTS-kSubS and BF-kSubS. In particular, we show that they vastly outperform their standard ("non-subgoal") counterparts. As a testing ground, we consider three challenging domains: Sokoban, Rubik's Cube, and INT. All of them require non-trivial reasoning. The Rubik's Cube is a well-known 3-D combination puzzle. Sokoban is a complex video puzzle game known to be NP-hard and thus challenging for planning methods. INT [55] is a recent theorem proving benchmark.

4.1 Training protocol and baselines

Our experiments consist of three stages. First, we collect domain-specific expert data, see Section 4.2. Secondly, we train the subgoal generator, low-level conditional policy, and value function networks using the data and targets described in Section 3. For more details see Appendix D. Eventually, we evaluate the planning performance of MCTS-kSubS and BF-kSubS, details of which are presented below. In the second step, we use supervised learning, which makes our setup stable with respect to network initialization, see details in Appendix D.1.3.

As baselines, we use BestFS and MCTS (being a single-player implementation of AlphaZero). In INT and Rubik's Cube, both the algorithms utilize policy networks (trained with behavioral cloning, on the same dataset, which we used to train kSubS). Note that distribution over actions induces a distribution over states; thus the policy network can be regarded as a subgoal generator for $k = 1$. More details about the baselines can be found in Appendix I.

4.2 Search Domains and Datasets

Sokoban is a single-player complex game in which a player controls an agent whose goal is to place boxes on target locations solely by pushing them; without crossing any obstacles or walls. Sokoban has recently been used to test the boundaries in RL [16, 29]. Sokoban is known to be hard [10], mainly due to its combinatorial complexity and the existence of irreversible states. Deciding if a given Sokoban board is solvable is an NP-hard problem [8].

We collect the expert dataset consisting of all successful trajectories occurring during the training of an MCTS agent (using an implementation of [29]). These are suboptimal, especially in the early phases of the training or for harder boards. For both expert training and kSubS evaluation, we generate Sokoban boards following the approach of [38].

Rubik's Cube is a classical 3-dimensional puzzle. It is considered challenging due to the fact that the search space has more than 4.3×10^{18} configurations. Similarly to Sokoban, Rubik's Cube has been recently used as a testing ground for RL methods [1].

To collect the expert dataset, we generate random paths of length 30 starting from the solved cube and take them backward. These backward solutions are highly sub-optimal (optimal solutions are proven to be shorter than 26 [39]).

INT: Inequality Benchmark for Theorem Proving. INT provides a generator of inequalities, which produces mathematical formulas along with their proofs, see [55, Section 3.3]. Proofs are represented as sequences of consecutively applied mathematical axioms (there are $K = 18$ axioms in total). An action in INT is a tuple containing an axiom and its input entities. The action space in this problem can be very large, reaching up to 10^6 elements, which significantly complicates planning.

The INT generator constructs paths by randomly applying axioms starting with a trivial statement. Such a path taken backward constitutes the proof of its final statement (not guaranteed to be optimal). The proof length, denoted by L, is an important hyperparameter regulating the difficulty – we use $5, 10, 15$.

For more details on datasets see Appendix C.

	INT	Sokoban	Rubik
k	3	4	4
C_1	400	5000	1500
C_2	4	4	7
C_3	4	4	3
C_4	1	0.98	1

Table 1: BF-kSubS hyperparameters.

4.3 Main results

In this section, we present our most important finding: Subgoal Search enables for more efficient search and consequently scales up to problems with higher difficulty. Specifically, MCTS-kSubS and BF-kSubS perform significantly better than the respective methods not using subgoals, including state-of-the-art on INT.

In Figure 1, we present the performance of Subgoal Search. We measure the *success rate* as a function of the *search budget*. The success rate is measured on 1000 instances of a given problem (which results in confidence intervals within ± 0.03). For BF-kSubS the search budget is referred to as *graph size* and includes the number of nodes visited by Algorithm 1. For INT and Rubik's Cube, we include both the subgoal generated by `SUB_GENERATE` and the nodes visited by `GET_PATH` (as they induce a significant computational cost stemming from using low-level policy π in Algorithm 2). For Sokoban, we use Algorithm 9 to realize `GET_PATH`, as it has a negligible cost (less than 1% of the total runtime of Algorithm 1), we do not include these nodes into graph size.

For MCTS, we report *MCTS passes*, which is a common metric for MCTS, see details in Appendix A.1.

Below we discuss the results separately for each domain. We provide examples of solutions and generated subgoals in Appendix H.

INT The difficulty of the problems in INT increases fast with the proof length L and the number of accessible axioms. W used $K = 18$; all of available axioms. We observe, that BF-kSubS scales to proofs of length $L = 10$ and $L = 15$, which are significantly harder than $L = 5$ considered in [55], see Table 2. The same holds for MCTS-kSubS, see Appendix A.2.

We check also that MCTS-kSubS vastly outperforms the baseline - AlphaZero algorithm, see Figure 1 (top, left). An MCTS-based agent was also evaluated in [55]. Its implementation uses graph neural

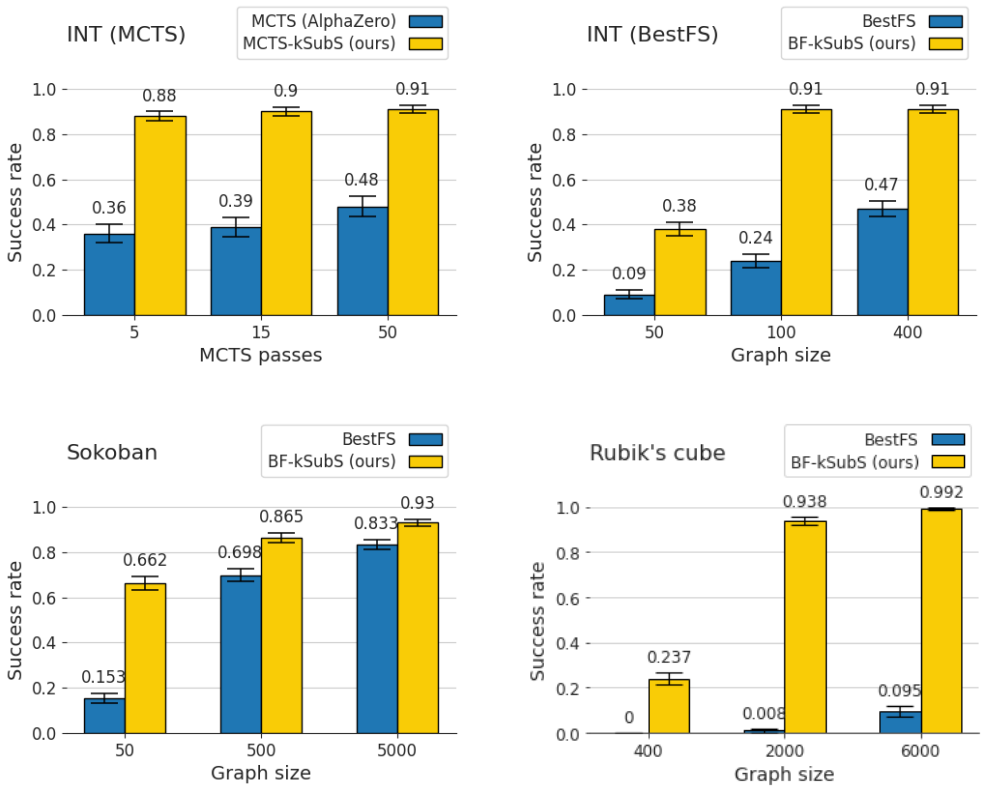

Figure 1: The performance of Subgoal Search. (top, left) comparison on INT (with the proof length 15) to AlphaZero. (top, right) BF-kSubS consistently achieves high performance even for small computational budgets. (bottom, left) similarly on Sokoban (board size 12x12 with 4 boxes) the advantage of BF-kSubS is clearly visible for small budget. (bottom, right) BestFS fails to solve Rubik's Cube, while BF-kSubS can achieve near-perfect performance.

	Proof length	5		10		15	
	Method	BestFS	BF-kSubS (ours)	BestFS	BF-kSubS (ours)	BestFS	BF-kSubS (ours)
Graph size	50	0.82	**0.99**	0.47	**0.97**	0.09	**0.38**
	100	0.89	**0.99**	0.64	**0.99**	0.24	**0.91**
	200	0.92	**0.99**	0.67	**0.99**	0.35	**0.91**
	400	0.93	**0.99**	0.72	**0.99**	0.47	**0.91**

Table 2: INT success rates for various proof lengths and graphs sizes.

networks architectures and achieves 92% success rate for $L = 5$. Our transformed-based baseline is stronger - it solves over 99% instances on the same problem set.

Sokoban Using BF-kSubS allows for significantly higher success rates rates within the same computational budget, see Table 3. Our solution scales well to the board size as big as 20×20; note that 10×10 boards are typically used in deep RL research [16, 38]. Importantly, we observe that already for a small computational budget (graph size 1000) BF-kSubS obtains higher success rates than the expert we used to create the dataset (these are 78%, 67%, 60% for the respective board sizes).

We also tested how the quality of BF-kSubS depends on the size of the training dataset for Sokoban, the results can be found in Appendix F.

Rubik's Cube BF-kSubS solves nearly 100% of cubes, BestFS solve less than 10%, see Figure 1 (bottom, right). This is perhaps the most striking example of the advantage of using a subgoal generator instead of low-level actions. We present possible explanation in Appendix K.

	Board size	12 x 12		16 x 16		20 x 20	
	Method	BestFS	BF-kSubS (ours)	BestFS	BF-kSubS (ours)	BestFS	BF-kSubS (ours)
Graph size	50	0.15	**0.66**	0.04	**0.42**	0.02	**0.46**
	100	0.46	**0.79**	0.23	**0.62**	0.10	**0.55**
	1000	0.75	**0.89**	0.61	**0.79**	0.47	**0.70**
	5000	0.83	**0.93**	0.69	**0.85**	0.58	**0.77**

Table 3: Sokoban success rates for various board sizes (each with 4 boxes).

Out-of-distribution (OOD) generalization OOD generalization is considered to be the crucial ability to make progress in hard combinatorial optimization problems [4] and automated theorem proving [55]. The INT inequality generator has been specifically designed to benchmark this phenomenon. We check that Subgoal Search trained on proofs on length 10 generalizes favourably to longer problems, see Figure 4. Following [55], we speculate that search is a computational mechanism that delivers OOD generalization.

It might be hard to compare computational budgets between various algorithms and their versions. In Appendix E we measure that BF-kSubS and MCTS-kSubS offer very practical benefits, sometimes as much as $7\times$ faster execution.

4.4 Analysis of k (subgoal distance) parameter

The subgoals are trained to predict states k steps ahead of the current one. Higher k should make planning easier as the search graph is smaller. However, as k increases, the quality of the generator may drop, and thus the overall effect is uncertain. Similarly, the task of the low-level conditional policy becomes more difficult as k increases. The optimal value of k is 3 and 4 for INT and Rubik's Cube, respectively. In these environments, increasing k further degrades performance. In Sokoban, we observe monotonic improvement up to $k = 10$. This is perhaps because low-level conditional policy (Algorithm 9, based on breadth-first search) never fails to fill the path from a state to the valid subgoal. The running cost of Algorithm 9 quickly becomes unacceptable (recall that for $k = 4$, which we used in the main experiment, it has still a negligible cost - below $< 1\%$ of the total runtime).

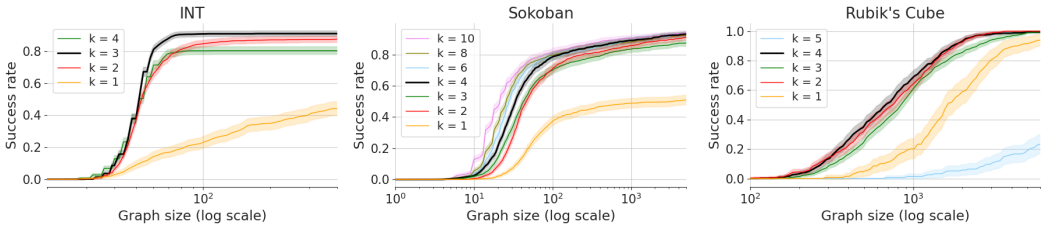

Figure 2: BF-kSubS success rates for different values of k. Black curves represent the values of k used in the main experiments (that is $k = 4$ for Rubik's Cube and Sokoban and $k = 3$ for INT).

4.5 Quality of subgoals

The learned subgoal generator is likely to be imperfect (especially in hard problems). We study this on 10×10 boards of Sokoban, which are small enough to calculate the true distance $dist$ to the solution using the Dijkstra algorithm. In Figure 3, we study $\Delta := dist_{s_1} - dist_{s_2}$, where s_1 is a sampled state and s_2 is a subgoal generated from s_1. Ideally, the histogram should concentrate on $k = 4$ used in training. We see that in slightly more than 65% of cases subgoals lead to an improvement.

The low-level conditional policy in Algorithm 2 provides additional verification of generated states. We check that in INT and Rubik's Cube, about 50% of generated subgoals can be reached by this policy (the rest is discarded).

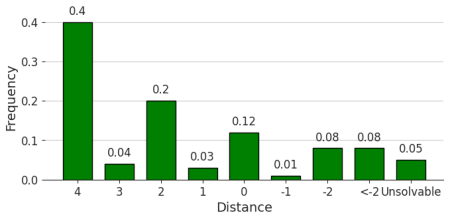

Figure 3: Histogram of Δ. Note that 17% of subgoals increases the distance. Additional, 5% leads to unsolvable "dead states" present in Sokoban.

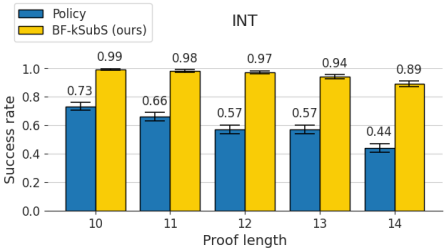

Figure 4: Out-of-distribution generalization to longer proofs. We compare with the behavioral cloning agent (Policy) studied in [55].

4.6 Value errors

There might be various explanations for the success of our method. One of them is that Subgoal Search better handles errors of learned value functions. In this section, we discuss this using a synthetic grid world example and performing statistical analysis on the real value function trained to approximate the distance to the solution (as described in Section 3).

Grid world example Consider a grid world with the state space $S = \{1, \ldots, n\}^m$, with $(0, \ldots, 0)$ being the initial state and $(n, \ldots, n)$ the goal state. A pair of states is connected by an edge if they are at distance 1 from each other. Let:

- Synthetic value function: the negative distance to the goal plus i.i.d Gaussian noise $\mathcal{N}(0, \sigma^2)$.
- Synthetic SUB_GENERATE (instead of Algorithm 3): Let $B_k(s)$ be the states within distance k from s. We return $C_3 - 1$ states sampled uniformly from $B_k(s)$ and a one "good subgoal" being a state in $B_k(s)$ with the minimal distance to the solution.
- Node expansion in BestFS (baseline): implemented as SUB_GENERATE above with $k = 1$.

In this setup, one easily sees that the probability that the good subgoal will have the highest value estimation among the generated states grows with k. Consequently, kSubS can handle higher levels of noise than the baseline BestFS, see Table 4.

σ	BestFS	BF-kSubS
3	0.999	1
10	0.142	1
20	0.006	0.983

Table 4: Success rates on the grid world ($m = 6, n = 10$), depending on the value function noise scale. We use the search budget of 500 nodes and $k = 4$ for kSubS.

Value monotonicity Imagine a solution path from the starting state to the goal state. Due to the errors, the value estimates on the path may not be monotonic. This is an undesirable property, which is likely to hinder search and make finding the path harder. Now consider the subpath consisting of consecutive states spaced k actions apart, as can be constructed by Subgoal Search. For this version, the value estimates are more likely to be monotonic and easier to find. To illustrate this, we measure monotonicity on solution paths found by our algorithm for INT. The probability that value decreases when moving k steps ahead drops from 0.32 when $k = 1$ to mere 0.02 for $k = 4$ (see Table 7 in Appendix).

Overoptimism Alternatively, one can consider that erroneously positive values misguide a search method (a phenomenon known as over-optimism [19]). To illustrate this, consider $\mathcal{S}_3(s)$, the set of all states having the same distance to the solution as s and within distance 3 from s. Intuitively, $\mathcal{S}_3(s)$ contains similar states with respect to the difficulty of solving. In Sokoban, the standard deviation of value function prediction for $\mathcal{S}_3(s)$ is equal to 2.43 (averaged over different s on Sokoban boards). This is high when compared to the average increase of value for moving one step closer to the solution, which is only 1.34. Consequently, it is likely that $\mathcal{S}_3(s)$ contains a suboptimal state, e.g., having a higher value than the best immediate neighbors of s (which by properties of the game will be closer to solution in Sokoban). Indeed, we measure that the probability of such an event is 64%. However, it drops significantly to 29% if one considers states closer by 4 steps (say given by a subgoal generator).

5 Limitations and future work

In this section, we list some limitations of our work and suggest further research directions.

Reliance on expert data In this version, we use expert data to train learnable models. As kSubS improves the performance, we speculate that training akin to AlphaZero can be used, i.e. in a planner-learner loop without any outside knowledge.

Optimality and completeness kSubS searches over a reduced state space, which might produce suboptimal solutions or even fail to find them. This is arguably unavoidable if we seek an efficient method for complex problems.

Subgoals definition We use simple k-step ahead subgoals, which is perhaps not always optimal. Our method can be coupled with other subgoal paradigms. Unsupervised detection of landmarks (see e.g. [56]) seems an attractive future research direction.

More environments In future work, we plan to test kSubS on more environments to understand its strengths and weaknesses better. In this work, we generate subgoals in the state space, which might be limiting for tasks with high dimensional input (e.g., visual).

Reliance on a model of the environment We use a perfect model of the environment, which is a common practice for some environments, e.g., INT. Extending kSubS to use learned (imperfect) models is an important future research direction.

Determinism Our method requires the environment to be deterministic.

OOD generalization A promising future direction is to investigate and leverage the out-of-distribution generalization delivered by our method and compare to (somewhat contradictory) findings of [17, 55].

Classical planning methods For many search problems, the state space can be represented in factored fashion (or such representation can be learned [3]). In such cases, the search can be greatly improved with width-based methods [26, 12]. It is an interesting research direction to combine kSubS with such methods.

6 Conclusions

We propose Subgoal Search, a search algorithm based on subgoal generator. We present two practical implementations MCTS-kSubS and BF-kSubS meant to be effective in complex domains requiring reasoning. We confirm that indeed our implementations excel in Sokoban, Rubik's Cube, and inequality benchmark INT. Interestingly, a simple k step ahead mechanism of generating subgoals backed up by transformer-based architectures performs surprisingly well. This evidence, let us hypothesize, that our methods (and related) can be further scaled up to even harder reasoning tasks.

Acknowledgments and Disclosure of Funding

The work of Konrad Czechowski, Tomasz Odrzygóźdź and Piotr Miłoś was supported by the Polish National Science Center grant UMO-2017/26/E/ST6/00622. This research was supported by the PL-Grid Infrastructure. Our experiments were managed using `https://neptune.ai`. We would like to thank the Neptune team for providing us access to the team version and technical support.

References

[1] Forest Agostinelli, Stephen McAleer, Alexander Shmakov, and Pierre Baldi. Solving the rubik's cube with deep reinforcement learning and search. Nature Machine Intelligence, 1(8):356–363, 2019.

[2] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 5048–5058, 2017.

[3] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. European Journal of Operational Research, 2020.

[5] 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.

[6] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In International Conference on Machine Learning, pages 1430–1440. PMLR, 2021.

[7] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors, Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992], pages 271–278. Morgan Kaufmann, 1992.

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

[9] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. arXiv preprint arXiv:1910.13395, 2019.

[10] Alan Fern, Roni Khardon, and Prasad Tadepalli. The first learning track of the international planning competition. Machine Learning, 84(1-2):81–107, 2011.

[11] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In International conference on machine learning, pages 1515–1528. PMLR, 2018.

[12] Guillem Frances, Miquel Ramírez Jávega, Nir Lipovetzky, and Hector Geffner. Purely declarative action descriptions are overrated: Classical planning with simulators. In IJCAI 2017. Twenty-Sixth International Joint Conference on Artificial Intelligence; 2017 Aug 19-25; Melbourne, Australia.[California]: IJCAI; 2017. p. 4294-301. International Joint Conferences on Artificial Intelligence Organization (IJCAI), 2017.

[13] Thomas Gabor, Jan Peter, Thomy Phan, Christian Meyer, and Claudia Linnhoff-Popien. Subgoal-based temporal abstraction in Monte-Carlo Tree Search. In IJCAI, pages 5562–5568, 2019.

[14] Wei Gao, David Hsu, Wee Sun Lee, Shengmei Shen, and Karthikk Subramanian. Intention-net: Integrating planning and deep learning for goal-directed autonomous navigation. In 1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings, volume 78 of Proceedings of Machine Learning Research, pages 185–194. PMLR, 2017.

[15] Timothy Gowers. The importance of mathematics. Springer-Verlag, 2000.

[16] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. In International Conference on Machine Learning, pages 2464–2473. PMLR, 2019.

[17] Jessica B. Hamrick, Abram L. Friesen, Feryal Behbahani, Arthur Guez, Fabio Viola, Sims Witherspoon, Thomas Anthony, Lars Holger Buesing, Petar Velickovic, and Theophane Weber. On the role of planning in model-based deep reinforcement learning. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[18] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. Neuron, 95(2):245–258, 2017.

[19] Hado Hasselt. Double q-learning. Advances in neural information processing systems, 23:2613–2621, 2010.

[20] Jeffrey R Hollerman, Leon Tremblay, and Wolfram Schultz. Involvement of basal ganglia and orbitofrontal cortex in goal-directed behavior. Progress in brain research, 126:193–215, 2000.

[21] Dinesh Jayaraman, Frederik Ebert, Alexei A. Efros, and Sergey Levine. Time-agnostic prediction: Predicting predictable video frames. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[22] Leslie Pack Kaelbling. Learning to achieve goals. In Ruzena Bajcsy, editor, Proceedings of the 13th International Joint Conference on Artificial Intelligence. Chambéry, France, August 28 - September 3, 1993, pages 1094–1099. Morgan Kaufmann, 1993.

[23] Junsu Kim, Younggyo Seo, and Jinwoo Shin. Landmark-guided subgoal generation in hierarchical reinforcement learning. In Thirty-Fifth Conference on Neural Information Processing Systems, 2021.

[24] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 11566–11575, 2019.

[25] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J. Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, pages 8747–8758, 2018.

[26] Nir Lipovetzky and Hector Geffner. Width and serialization of classical planning problems. In ECAI 2012, pages 540–545. IOS Press, 2012.

[27] Kara Liu, Thanard Kurutach, Christine Tung, Pieter Abbeel, and Aviv Tamar. Hallucinative topological memory for zero-shot visual planning. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 6259–6270. PMLR, 2020.

[28] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. CoRR, abs/2001.08210, 2020.

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

[30] Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[31] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 14814–14825, 2019.

[32] Giambattista Parascandolo, Lars Buesing, Josh Merel, Leonard Hasenclever, John Aslanides, Jessica B Hamrick, Nicolas Heess, Alexander Neitz, and Theophane Weber. Divide-and-conquer monte carlo tree search for goal-directed planning. arXiv preprint arXiv:2004.11410, 2020.

[33] Sujoy Paul, Jeroen Vanbaar, and Amit Roy-Chowdhury. Learning from trajectories via subgoal discovery. Advances in Neural Information Processing Systems, 32:8411–8421, 2019.

[34] Karl Pertsch, Oleh Rybkin, Frederik Ebert, Shenghao Zhou, Dinesh Jayaraman, Chelsea Finn, and Sergey Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

[35] Karl Pertsch, Oleh Rybkin, Jingyun Yang, Shenghao Zhou, Konstantinos G. Derpanis, Kostas Daniilidis, Joseph J. Lim, and Andrew Jaegle. Keyframing the future: Keyframe discovery for visual prediction and planning. In Alexandre M. Bayen, Ali Jadbabaie, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020, volume 120 of Proceedings of Machine Learning Research, pages 969–979. PMLR, 2020.

[36] Silviu Pitis, Harris Chan, Stephen Zhao, Bradly C. Stadie, and Jimmy Ba. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of Proceedings of Machine Learning Research, pages 7750–7761. PMLR, 2020.

[37] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393, 2020.

[38] 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.

[39] Tomas Rokicki and M Davidson. God's number is 26 in the quarter-turn metric, 2014.

[40] Stuart Russell and Peter Norvig. Artificial intelligence: A modern approach. ed. 3. 2010.

[41] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net, 2018.

[42] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489, 2016.

[43] 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.

[44] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. ArXiv, abs/1712.01815, 2017.

[45] 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.

[46] Gregory J. Stein, Christopher Bradley, and Nicholas Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In 2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings, volume 87 of Proceedings of Machine Learning Research, pages 213–222. PMLR, 2018.

[47] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In Liz Sonenberg, Peter Stone, Kagan Tumer, and Pinar Yolum, editors, 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3, pages 761–768. IFAAMAS, 2011.

[48] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artif. Intell., 112(1-2):181–211, 1999.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman

Garnett, editors, Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[50] Vivek Veeriah, Tom Zahavy, Matteo Hessel, Zhongwen Xu, Junhyuk Oh, Iurii Kemaev, Hado van Hasselt, David Silver, and Satinder Singh. Discovery of options via meta-learned subgoals. arXiv preprint arXiv:2102.06741, 2021.

[51] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Manfred Otto Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. ArXiv, abs/1703.01161, 2017.

[52] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, Nov 2019.

[53] David Wilkins. Using patterns and plans in chess. Artif. Intell., 14(2):165–203, 1980.

[54] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. CoRR, abs/1910.03771, 2019.

[55] Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger Grosse. Int: An inequality benchmark for evaluating generalization in theorem proving. arXiv preprint arXiv:2007.02924, 2020.

[56] Lunjun Zhang, Ge Yang, and Bradly C. Stadie. World model as a graph: Learning latent landmarks for planning. CoRR, abs/2011.12491, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] In the Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In Section **??**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the URL to the code repository in Section 1.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix sections B, D and C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In Appendix L
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] For one of domains we generate expert data with implementation of MCTS agent published with work [29], as noted in Section 4.2.

(b) Did you mention the license of the assets? [N/A] The implementation noted above is made available on github, without any licence.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We release our code, the URL is in Section 1.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]