Crafting a Pogo Stick in Minecraft with Heuristic Search

Yarin Benyamin[1], **Argaman Mordoch**[1], **Shahaf Shperberg**[1], **Wiktor Piotrowski**[2], **Roni Stern**[1]

[1]Ben-Gurion University of the Negev
[2]Palo Alto Research Cener, SRI
bnyamin@post.bgu.ac.il, mordocha@post.bgu.ac.il, shperbsh@bgu.ac.il, wiktor.piotrowski@sri.com, roni.stern@gmail.com

Abstract

In this work, we tackle a challenging combinatorial search task within the immersive world of *Minecraft*, a sandbox game renowned for its intricate environment. Specifically, we focus on a task where an agent must craft a wooden pogo stick by efficiently gathering resources and crafting items according to predefined recipes. This problem is naturally modeled as a numeric planning problem, and we show that it serves as a formidable testbed for state-of-the-art planners. To tackle this task effectively, we propose measures for quantifying how often an action has been used to generate states that were expanded during the search, and how often actions that are applicable in generated states have been previously employed. Based on these notions of *action novelty*, we develop several domain-independent heuristic functions and integrate them into a numeric planner. Our results showcase that using our action novelty heuristics allows the planner to solve significantly more problem instances and scale to larger maps than state-of-the-art numeric planners.

Introduction

Minecraft is an extremely popular sandbox video game. Its rich and complex environment provides many opportunities for AI agents to learn and engage with the game. The game's open-ended design allows the creation of unique tasks and challenges for the agents, providing a broad spectrum for researchers to experiment with different AI techniques and applications. Indeed, various Minecraft tasks have been posed as an AI challenge (Guss et al. 2019; Goss et al. 2023).

Most AI research on Minecraft focused on either applying Reinforcement Learning (RL) to solve the problem (Tessler et al. 2017), learning an action model for planning (James, Rosman, and Konidaris 2020; Benyamin et al. 2023), or modeling the problem for a domain-independent planner (Roberts et al. 2017). RL approaches require millions of interactions with the environment to succeed, often supplemented by examples of expert solutions to the same problem.

In this work, we focus on the combinatorial search challenge of solving a specific Minecraft task, Craft Wooden Pogo. We describe this task and show that it poses a significant challenge to state-of-the-art domain-independent planners. Then, we propose several heuristics for solving this problem, which can be used within a domain-independent planner. These heuristics are designed to guide the search towards exploring actions that were rarely used before.

We evaluate these heuristics experimentally on a range of problems within our Minecraft domain, showing that when integrated with a standard domain-independent planner, our heuristics outperform state-of-the-art planners such as Metric FF (Hoffmann 2003) and ENHSP (Scala et al. 2017). While our heuristics were designed for solving the Craft Wooden Pogo task, they are domain-independent and can be used in other domains as well.

Background

Planning problems in domains with deterministic action outcomes and fully observable states, described with discrete and continuous state variables, can be defined using the Planning Domain Definition Language (PDDL) (Ghallab et al. 1998), specifically, PDDL2.1 (Fox and Long 2003). PDDL2.1 is very expressive. For our purposes, we consider the following subset of PDDL2.1. A planning domain is defined by a tuple $D = \langle F, X, A \rangle$ where F is a finite set of Boolean variables, X is a set of numeric variables, and A is a set of actions. A state is an assignment of values to all variables in $F \cup X$. Every action $a \in A$ is defined by a tuple $\langle \mathit{name}(a), \mathit{pre}(a), \mathit{eff}(a) \rangle$ representing the action's name, preconditions, and effects, respectively. Preconditions are assignments over the Boolean variables and conditions over the numeric variables, specifying when the action can be applied. The effects of an action are a set of assignments over F and X, representing how the state changes after applying a. A planning problem in PDDL is defined by $\langle D, s_0, G \rangle$ where D is a domain, and s_0 is the initial state. G is the goal, represented by a set of assignments of values to a subset of the Boolean variables and a set of conditions over the numeric variables. A solution to a planning problem is a *plan*, i.e., a sequence of actions applicable in s_0 and resulting in a state s_G in which G is satisfied.

Planning problems and domains are often specified in a *lifted* manner. This means a planning domain defines parameterized predicates and functions instead of Boolean and numeric variables, and the actions are also parameterized. A planning problem defines a set of objects that can be used as parameters for these predicates, functions, and actions. For example, in our Minecraft domain the location of the

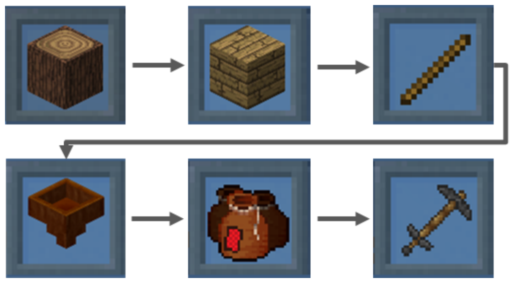

Figure 1: A plan to accomplish the Craft Wooden Pogo task.

agent is a parameterized predicate $(at?cell)$, and the problem specifies an object for each cell in the grid.

Polycraft

Polycraft (Palucka 2017) is an interface to Minecraft, part of the Polycraft World AI Lab (PAL) (Goss et al. 2023)[1]. PAL provides an API for AI agents to interact with Minecraft's environment, allowing commands such as "Move", "Mine resources", and "Craft item" to be sent to the game's main character, each with pre-defined preconditions, effects, and costs. In addition, PAL supports *symbolic* observations of the current state, making it ideal for planning algorithms, which require a symbolic model of the environment for problem-solving. This property differentiates Polycraft from other Minecraft research frameworks such as MineRL (Guss et al. 2019), which provides a visual, pixel-based representation of the game. Thus, in this work, we used PAL to interact with the Minecraft environment.

State and Action Novelty

Lipovetzky and Geffner (2012) introduced the concept of novelty in classical planning to gauge the uniqueness of a state s by comparing its content with that of previously visited states. Several search algorithms, such as Iterated Width and Best-First Width-Search (Lipovetzky and Geffner 2017) use the novelty of generated states to guide the search. More recently, Lei et al. (2023) have explored the notion of *action novelty* in the realm of generalized planning, where structured plans (potentially containing loops and conditionals) are devised to solve a set of problem instances within a given domain. None of these prior works deal with numeric planning problems as we do. Moreover, they consider either novelty w.r.t the state variables or w.r.t the number of times an action appears in a (generalized) plan. Our approach to novelty refers to the number of times an action has been used to generate states in the search space explored so far. We discuss below why existing novelty-based searches that rely solely on the novelty of the state variables are not expected to work well in our domain. Our work is different from diverse planning, where the objective is to find a set of plans that are different enough from each other (Katz and Sohrabi 2020).

Problem Definition

This work focuses on the Craft Wooden Pogo task, as defined in the PAL Minecraft environment (Goss et al. 2023). In this task, the Minecraft agent, colloquially called Steve, is located in a field comprising $N \times N$ blocks and surrounded by unbreakable bedrock walls. The field includes multiple trees and a crafting table. Steve is tasked with crafting a pogo stick, which requires performing the following actions (illustrated in Figure 1):

1. Harvest at least three wood blocks from trees.
2. Use the wood to craft planks.
3. Use planks to craft sticks.
4. Use some of the sticks and planks to craft a tree tap
5. Place a tree tap near a tree to collect polyisoprene sacks.
6. Use the remaining sticks, planks, and polyisoprene sacks to craft a wooden Pogo stick.

The agent observes the entire map, including details like cell types, inventory contents (type and quantity of items), and its position. The available actions in PAL are:

1. **TP_TO** - teleport from the current location to another cell on the map.
2. **BREAK**- break a tree and add the resulting log to the inventory. This can only be done when in a tree cell.
3. **CRAFT_PLANK** - craft 4 planks using a single log from the inventory.
4. **CRAFT_STICK** - craft 4 sticks using 2 planks from the inventory.
5. **CRAFT_TREE_TAP** - teleport to the crafting table, craft a tree tap using 5 planks and 1 stick from the inventory.
6. **PLACE_TREE_TAP** - place the tree tap on a tree and collect a polyisoprene sack. This action can only be done if the inventory includes at least one tree tap and the agent is in front of a tree.
7. **CRAFT_WOODEN_POGO** - teleport to the crafting table, and craft a wooden pogo stick using a polyisoprene sack, 2 planks, and 4 sticks from the inventory.

The Craft Wooden Pogo can be directly modeled in PDDL2.1. For example, the preconditions of *CRAFT_STICK* are to have at least 2 planks in the inventory, and the effects are to decrease two planks and add one stick to the agent's inventory. Other ways to model the Craft Wooden Pogo task in PDDL also exist.

The original task in PAL was predefined with a fixed map size and predetermined positions for the trees, the crafting table, and the agent. Moreover, the agent always starts with an empty inventory. To introduce variability across problem instances, we developed a problem generator that generates initial states in which it randomly assigns (1) the agent's starting position on the map, (2) the quantity and placement of trees, (3) the items present in the agent's inventory, and (4) the crafting table's position. Thus, different sequences of actions are needed to solve different problem instances.

[1]https://github.com/PolycraftWorld/PAL

Note that the *TP* action includes two parameters: the current position of the agent and its target position. Similarly, actions such as *BREAK*, *CRAFT_TREE_TAP*, *PLACE_TREE_TAP*, and *CRAFT_WOODEN_POGO* also require the current position of the agent. Consequently, the total number of grounded actions is $O(N^4)$ where N represents the length and width of the map. The number of states is also formidable since a state includes the location of the character, the number of items of each type currently in the inventory, and the state of the trees in the map (i.e., which tree has been broken already). Thus, this problem presents a significant challenge for planners due to its large action and state spaces.

Action-Based Novelty Heuristics

Most domain-independent planners capable of solving PDDL2.1 problems, such as ours, are based on heuristic search. Yet, existing heuristics fail to guide the search effectively in the Craft Wooden Pogo domain on large maps, as demonstrated experimentally in Section . Novelty-based search algorithms (Lipovetzky and Geffner 2017) that rely only on features of the state are also not expected to be effective in our domain, since they have no way to prefer moving to one grid cell over the other. In this section, we propose several heuristics that enable scaling to significantly larger problems. These heuristics prioritize nodes in the search tree based on the novelty of the actions that lead to them and the novelty of the actions they enable. Importantly, all the proposed heuristics are domain-independent, in the sense that they do not include any Minecraft-specific information.

To define our heuristics, we associate every search node n generated during the search with a tuple (g, h, a, s), where g denotes the cost of reaching n from the start state, h represents the heuristic value of n, a signifies the (lifted) action used to generate n, and s indicates the state represented by n. We denote the set of nodes expanded during the search as CLOSED, also known as the *closed list*.

The first heuristic we consider prioritizes nodes that enable more actions to be applied. Formally, given a node $n = (g, h, a, s)$ this heuristic is computed as follows:

$$h_{AA}(s) = \frac{1}{\text{AA}(s)}$$

where $AA(s)$ is the number of actions (lifted) applicable in state s. The h_{AA} heuristic is agnostic to CLOSED, depending only on the state the search node represents. The other heuristics analyze CLOSED and consider the following measures of *action novelty*.

Definition 1 (Action Novelties)**.** *Given a node $n = (g, h, a, s)$ and CLOSED, we define the Explored-Action Novelty (E-AN) and Applicable-Action Novelty (A-AN) as:*

$$\textit{E-AN}(n, \textit{CLOSED}) = |\{a = a' | (g', h', a', s') \in \textit{CLOSED}\}|$$

$$\textit{A-AN}(n, \textit{CLOSED}) = \frac{1}{\sum_{a \in AA(s)} \frac{1}{|\{a=a'|(g',h',a',s') \in \textit{CLOSED}\}|}}$$

where $AA(s)$ is the list of applicable (lifted) actions in s.

The Explored-Action Novelty of a node n reflects how often the action leading to n was used so far to generate nodes that were expanded during the search. The Applicable-Action Novelty of a node n is based on the number of times the actions that are applicable in n were used so far to generate nodes that were expanded during the search.

Based on these definitions of Action Novelty, we introduce the following heuristic functions.

$$h_{\text{E-AN}}(n, \text{CLOSED}) = \text{E-AN}(n, \text{CLOSED})$$

$$h_{\text{A-AN}}(n, \text{CLOSED}) = \text{A-AN}(n, \text{CLOSED})$$

where $h_{\text{E-AN}}$ relies on the Explored-Action Novelty and $h_{\text{A-AN}}$ relies on the Applicable-Action Novelty.

Lastly, we introduce a combined heuristic that integrates both Explored- and Applicable-based action novelty:

$$h_{\text{EA-AN}}(n, \text{CLOSED}) = \text{E-AN}(n, \text{CLOSED}) + \text{A-AN}(n, \text{CLOSED})$$

this heuristic aims to balance between the novelty of the action leading to a node and the novelty of applicable actions from that node. We explored other combinations of E-AN and A-AN, and this sum worked best in our experiments.

Other than h_{AA}, all proposed heuristics depend on CLOSED. This entails: (i) processing CLOSED whenever a node is generated, and (ii) re-computing the heuristic for all nodes after each expansion. To address (i), we recognize that the only relevant information from the closed list is the frequency of each action's application. Hence, we maintained a counter for each action. These counters, implemented as a dictionary of counters, indicate the number of times each action has been applied up to the current state of the search. To mitigate (ii), we implemented a lazy evaluation of the heuristic values similar to the Lazy A* algorithm (Karpas et al. 2018). When evaluating a node for expansion, we recalculate its heuristic value. If the recalculated heuristic value is greater than its previous value, we delay the expansion of the node, re-prioritizing it based on the updated value.

Experimental Results

We conducted experiments on Craft Wooden Pogo problems with maps of size 6×6, 10×10, 15×15, 30×30, and 45×45. For every map size, we created 50 problems, which differ in the initial content of the inventory and the number of trees in the map. Specifically, the number of items in the inventory initially ranged from zero to eight for all items except for the polyisoprene sack and the Pogo stick, which were always zero (otherwise the problem is not challenging). We set the number of trees on each map to range from zero to (map size)/3. The machines used for the experiments consist of two Intel Xeon E5-2620 processors and 48GB of RAM. Each algorithm ran with a time limit of 30 min.

We implemented our heuristics into NYX planner (Piotrowski and Perez 2024), a versatile, Python-based, domain-independent planner. We chose Nyx because it enables easy integration of new heuristics into the search process. As a baseline, we run experiments with two blind searches, namely Depth-First Search (DFS) and Breadth-First Search (BFS). Then, we run experiments with Greedy Best-First Search (GBFS) using each of our newly introduced heuristics. We also experimented with using A* instead of GBFS, but preliminary results showed GBFS worked better. This is reasonable as we do not aim to find optimal solutions and align with our approach, as our heuristic differs from the conventional measure of distance to the goal; instead, we prioritize based on the novelty score of the state.

We also experimented with two state-of-the-art numeric planners: Metric-FF (Hoffmann 2003) and ENHSP (Scala et al. 2016). Metric-FF was tested with three primary configurations: prioritizing cost minimization using variants of the A* algorithm, employing BFS with heuristic-based approaches, or combining Enforced Hill Climbing (EHC) with BFS (Standard-FF). Among these, the configuration that yielded the most promising results was Standard-FF.

ENHSP configurations encompassed various combinations of heuristics and search algorithms, including Greedy Best First Search, A*, and numeric heuristics like MRP, h_{add}, AIBR, h_{max}, and landmark heuristics. Our experiments revealed that utilizing A* with the h_{max} heuristic outperformed other configurations. We report only the best configuration for each baseline.

Results and Trends

Map Size	6×6	10×10	15×15	30×30	45×45
NYX$_{\text{BFS}}$	50 (14)	29 (67)	9 (219)	-	-
NYX$_{\text{DFS}}$	50 (17)	50 (35)	44 (159)	-	-
NYX$_{\text{AA}}$	50 (14)	50 (45)	24 (141)	-	-
NYX$_{\text{E-AN}}$	45 (120)	7 (205)	-	-	-
NYX$_{\text{A-AN}}$	50 (18)	37 (457)	5 (906)	-	-
NYX$_{\text{EA-AN}}$	50 (5)	50 (5)	50 (10)	49 (73)	50 (748)
ENHSP	50 (1)	50 (32)	45 (170)	13 (1434)	-
MetricFF	50 (0)	50 (1)	50 (5)	-	-

Table 1: Number of solved problems and runtime in seconds (appears in brackets) for different map sizes and algorithms.

Table 1 presents the number of problems solved and the average runtime (in brackets) by the different algorithms for maps of different sizes. Cells with “-” indicate the designated algorithm could not solve any problems for this map size. Consider first the results of Nyx when using BFS, DFS, and our four heuristics (the first 6 lines in Table 1). Neither $h_{\text{A-AN}}$ nor $h_{\text{E-AN}}$ exhibited improvements over the blind search variants, solving overall fewer problems. However, when combined into $h_{\text{EA-AN}}$, they collectively achieve the best results among all Nyx configurations. Furthermore, Nyx equipped with $h_{\text{EA-AN}}$ outperforms ENHSP across most of the map sizes. When comparing Nyx with $h_{\text{EA-AN}}$ to Metric-FF, we observe similar performance on the three smallest maps. However, upon scaling to larger maps, Metric-FF fails to solve any problems due to the substantial memory required for heuristic construction, which exceeds our system’s capabilities. Overall, Nyx equipped with $h_{\text{EA-AN}}$ exhibited superior performance, demonstrating significantly improved scalability.

Table 2 shows the average number of nodes expanded for Nyx with BFS, DFS, and GBFS with all the heuristics we proposed. The results show that, as expected, the number of nodes expanded by Nyx with $h_{\text{EA-AN}}$ is significantly smaller than all other cases. Interestingly, using the components of $h_{\text{EA-AN}}$ individually, i.e., using either $h_{\text{E-AN}}$ or $h_{\text{A-AN}}$, yields unimpressive results. For example, for 10×10 maps using $h_{\text{E-AN}}$ and $h_{\text{A-AN}}$ required expanding on average 257,726 and 670,296 nodes, respectively, while using their combination ($h_{\text{EA-AN}}$) resulted in solving more problems and expanded on average only 1,086 nodes.

Map Size	6×6	10×10	15×15	30×30	45×45
NYX$_{\text{BFS}}$	118,665	2,130,514	4,231,802	-	-
NYX$_{\text{DFS}}$	65,857	348,076	1,821,880	-	-
NYX$_{\text{AA}}$	17,247	838,806	2,057,879	-	-
NYX$_{\text{E-AN}}$	72,100	257,726	-	-	-
NYX$_{\text{A-AN}}$	19,763	670,296	387,674	-	-
NYX$_{\text{EA-AN}}$	443	1,086	2,525	10,212	23,367

Table 2: Avg. number of expanded states for different map sizes and Nyx configurations.

Related Works

The most widely adopted method for playing Minecraft is the Hierarchical Deep Reinforcement Learning Network (H-DRLN) (Tessler et al. 2017). This approach enables the agent to continuously learn multiple policies and adapt to new challenges within the game. The H-DRLN leverages a deep neural network to model the policy and value functions, resulting in high effectiveness across a variety of Minecraft tasks such as navigation, mining, and combat. Despite success, this Reinforcement Learning (RL) approach requires extensive training and environment interaction, while not guaranteeing feasible plans, making it less suitable for our context.

In the realm of planning, Wichlacz et al. (2019) used PDDL modeling to solve complex construction tasks in Minecraft. They modeled house-construction tasks as classical and as Hierarchical Task Network (HTN) (Georgievski and Aiello 2015) planning problems. They observed that even simple tasks present difficulties to current planners as the size of the world increases. Learning HTN domains from observations is an open problem. Importantly, there are key distinctions between the work of Wichlacz, Torralba, and Hoffmann (2019) and our own. First, they address different tasks (house construction vs crafting a pogo stick). Moreover, Wichlacz et al.’s work focused on problem modeling, whereas our research is primarily concerned with tackling the combinatorial search challenges inherent in problem-solving and scalability.

Conclusions and Future Work

In this work, we introduced the crafting of a wooden pogo stick in Minecraft as a planning problem, demonstrating its complexity for numeric planners. Moreover, we proposed domain-independent action-novelty-based approaches that notably enhance planners’ efficacy in solving this task, enabling scaling to large maps with numerous objects. The action-novelty heuristics we proposed are domain-independent, in the sense that they do not include any

Minecraft-specific element. We performed a limited evaluation of Nyx with $h_{\text{EA-AN}}$, the most effective heuristic for the Craft Wooden Pogo problem, on problems from two domains from the International Planning Competition (IPC) for numeric planning (McDermott 2000): Rover and Satellite. The results were disappointing: Nyx with $h_{\text{EA-AN}}$ was only able to solve 1 and 34 problems in Rovers and Satellites, respectively, within a 30-minute time limit. By contrast, ENHSP solved 3 and 0 problems from these domains, and Metric-FF solved 11 and 162 problems from these domains.

We conjecture that this is because our action-novelty approach lacks any goal-oriented element. Moreover, our heuristics are not expected to work well in scenarios where plans involve frequent repetition of the same action or in environments with a vast action space, most of which are irrelevant. An exciting direction for future research is to integrate the action-novelty scheme with goal-oriented heuristics and state novelty schemes (Lipovetzky and Geffner 2017) to achieve a better balance between exploring new actions, novelty of states, and progress towards the goal.

References

Benyamin, Y.; Mordoch, A.; Shperberg, S. S.; and Stern, R. 2023. Model Learning to Solve Minecraft Tasks. In *PRL Workshop in ICAPS*.

Fox, M.; and Long, D. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20: 61–124.

Georgievski, I.; and Aiello, M. 2015. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222: 124–156.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – the planning domain definition language. *AIPS Technical Report, Tech. Rep.*

Goss, S. A.; Steininger, R. J.; Narayanan, D.; Olivença, D. V.; Sun, Y.; Qiu, P.; Amato, J.; Voit, E. O.; Voit, W. E.; and Kildebeck, E. J. 2023. Polycraft World AI Lab (PAL): An Extensible Platform for Evaluating Artificial Intelligence Agents. *arXiv preprint arXiv:2301.11891*.

Guss, W. H.; Houghton, B.; Topin, N.; Wang, P.; Codel, C.; Veloso, M.; and Salakhutdinov, R. 2019. Minerl: A large-scale dataset of minecraft demonstrations. *arXiv preprint arXiv:1907.13440*.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research*, 20: 291–341.

James, S.; Rosman, B.; and Konidaris, G. 2020. Learning Object-Centric Representations for High-Level Planning in Minecraft. In *Workshop on Object-Oriented Learning at ICML*.

Karpas, E.; Betzalel, O.; Shimony, S. E.; Tolpin, D.; and Felner, A. 2018. Rational deployment of multiple heuristics in optimal state-space search. *Artificial Intelligence*, 256: 181–210.

Katz, M.; and Sohrabi, S. 2020. Reshaping diverse planning. In *AAAI Conference on Artificial Intelligence*, volume 34, 9892–9899.

Lei, C.; Lipovetzky, N.; and Ehinger, K. A. 2023. Novelty and Lifted Helpful Actions in Generalized Planning. In *SOCS*, 148–152. AAAI Press.

Lipovetzky, N.; and Geffner, H. 2012. Width and Serialization of Classical Planning Problems. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 540–545. IOS Press.

Lipovetzky, N.; and Geffner, H. 2017. Best-first width search: Exploration and exploitation in classical planning. In *AAAI Conference on Artificial Intelligence*, volume 31.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 13.

Palucka, T. 2017. Polycraft World teaches science through an endlessly expansive universe of virtual gaming: https://polycraft.utdallas.edu. *MRS Bulletin*, 42(1): 15–17.

Piotrowski, W.; and Perez, A. 2024. Real-World Planning with PDDL+ and Beyond. *arXiv preprint arXiv:2402.11901*.

Roberts, M.; Piotrowski, W.; Bevan, P.; Aha, D.; Fox, M.; Long, D.; and Magazzeni, D. 2017. Automated planning with goal reasoning in Minecraft. In *ICAPS workshop on Integrated Execution of Planning and Acting (IntEx)*.

Scala, E.; Haslum, P.; Magazzeni, D.; Thiébaux, S.; et al. 2017. Landmarks for Numeric Planning Problems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4384–4390.

Scala, E.; Haslum, P.; Thiébaux, S.; and Ramirez, M. 2016. Interval-based relaxation for general numeric planning. In *European Conference on Artificial Intelligence (ECAI)*, 655–663.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI conference on artificial intelligence*, volume 31.

Wichlacz, J.; Torralba, A.; and Hoffmann, J. 2019. Construction-planning models in minecraft. In *Proceedings of the ICAPS Workshop on Hierarchical Planning*, 1–5.