WHAT MAKES MATH PROBLEMS HARD FOR REINFORCEMENT LEARNING: A CASE STUDY

A. SHEHPER, A. MEDINA-MARDONES, B. LEWANDOWSKI, A. GRUEN, P. KUCHARSKI, AND S. GUKOV

Abstract. Using a long-standing conjecture from combinatorial group theory, we explore, from multiple angles, the challenges of finding rare instances carrying disproportionately high rewards. Based on lessons learned in the mathematical context defined by the Andrews–Curtis conjecture, we propose algorithmic improvements that can be relevant in other domains with ultra-sparse reward problems. Although our case study can be formulated as a game, its shortest winning sequences are potentially 10^6 or 10^9 times longer than those encountered in chess. In the process of our study, we demonstrate that one of the potential counterexamples due to Akbulut and Kirby, whose status escaped direct mathematical methods for 39 years, is stably AC-trivial.

Contents

Date: August 29, 2024.
2020 *Mathematics Subject Classification.* 68T09, 62R07, 55N31, 62R40.
Key words and phrases. reinforcement learning, large language models, topological data analysis, automated reasoning, search algorithms.

1. Introduction

We live in an extraordinary era where artificial intelligence (AI) is transforming numerous sectors and professions. Recent advancements in Large Language Models (LLMs) have empowered AI to read, write, and converse with a proficiency comparable to that of human experts. In the realm of board games, AI has outperformed even the most skilled human players, and it has tackled complex scientific challenges like protein folding, where steady progress was suddenly overtaken by a near-complete solution. As AI continues to evolve, one critical question remains: How wide is the range of domains in which AI systems can reason as effectively as humans?

Mathematics appears to be a natural progression on the path toward Artificial General Intelligence (AGI) due to its universal syntactic and logical structure, similar to that of natural language. Additionally, mathematics provides a framework for the quantitative evaluation of logical and analytical reasoning, making it an ideal domain for self-improving AI systems on the path to AGI. In a moment, we will explain another reason why mathematics could play a crucial role in AGI development, but first, we need to introduce one more key element: reinforcement learning (RL).

Machine learning, a subfield of AI, involves developing algorithms and statistical models that enable computers to learn from data and make predictions. Among the three primary areas of machine learning—supervised learning, unsupervised learning, and reinforcement learning—RL emphasizes learning through interaction with an environment and receiving feedback in the form of rewards or penalties. This aspect of machine learning, often characterized by its focus on AI models 'playing games,' will be central to our discussion.

A typical chess game lasts about 30 to 40 moves, with the longest recorded professional game reaching 269 moves, ending in a draw between Ivan Nikolic and Goran Arsovic in 1989. Notably, the number of moves in a typical chess game is relatively consistent, with the longest professional game having only about an order of magnitude more moves than the average. Similarly, a typical game of Go involves a few hundred moves, with the longest recorded professional game, played by Go Seigen and Kitani Minoru in 1933, lasting 411 moves.

At first glance, proving or disproving mathematical conjectures can be formulated as games. For example, proving a theorem involves finding a path from the hypothesis to the conclusion, consisting of basic logical steps, such as Lean steps. From the RL

perspective, this type of game can be quite complex due to its large action space. Conversely, finding examples or counterexamples to settle important conjectures may require only a few basic moves (actions); the case study in this paper serves as a good illustration of such a problem. In all cases, the problem is fundamentally a search process, with complexity largely determined by the size of the action space and the search space.

In addition, with hard research-level mathematics problems, one faces yet another challenge: the sought-after instance can be so rare and difficult to find that the problem effectively becomes a search for a needle in a haystack, i.e., a problem with ultra-sparse rewards. For example, in the context of theorem proving, one might consider an extremely hard theorem[1] that may require a very large number of steps. If there aren't many alternative proofs, finding even a small number of very long ones then becomes akin to searching for a needle in a haystack or, depending on one's preference, a search for a unicorn.[2]

Fortunately, mathematics offers a robust framework for developing and testing new algorithms with adaptive capabilities that dynamically 'learn how to learn.' Testing these algorithms on mathematical problems, rather than directly on societal issues like market crash predictions or extreme weather events, provides a risk-free and cost-effective approach. Additionally, this method offers the dual benefit of potentially solving hard mathematical problems and resolving long-standing conjectures in the process.

Our approach to problems of this type involves equipping the RL model with the ability to assess the hardness of problems during the training process. First and foremost, this requires a practically useful notion of hardness, a concept we thoroughly explore. By learning the distribution of problems based on their difficulty, one can enhance existing off-the-shelf algorithms with new self-improvement strategies, identifying key features that facilitate solving the most challenging problems.

In this paper, we begin a systematic implementation of this approach by carefully analyzing the distribution of hardness in potential counterexamples to a long-standing conjecture, the Andrews–Curtis conjecture. As with many other challenging problems, a natural measure of hardness is the number of steps an RL agent needs to take. What makes the Andrews–Curtis conjecture particularly suitable for our study is that it includes numerous examples requiring a hyper-exponential number of steps, providing an effective testbed for exploring the aforementioned questions through the lens of RL, search algorithms, and topological data analysis.

We should emphasize that this entire project was carried out using relatively modest computational resources that a small academic research group can easily afford. Consequently, we placed greater emphasis on theoretical tools and techniques that, when combined with scaled-up computational resources, can lead to further advancements.

---

[1] A proxy for such a problem could be the Riemann Hypothesis or any other unsolved Millennium Prize Problem.

[2] Similar challenges, though not as critical, also arise in non-research-level math problems; see, e.g., [PG23; DU24; Tri+24] for recent discussion. Meanwhile, in a parallel line of development, new benchmarks have been proposed in the past couple of years [Cob+20; OAC23], which can be useful in such contexts.

While our primary focus is on exploring the distribution of hardness with an eye toward algorithm development, we also resolve a particularly intriguing open case that has eluded direct mathematical approaches for decades:

Theorem 1. *The following potential counterexample introduced by Akbulut and Kirby [AK85] is stably AC-trivial:*

$$AK(3) = \langle x, y \mid x^3 = y^4, xyx = yxy \rangle.$$

The proof of this theorem is presented in Subsection 3.5. The rest of the paper is organized as follows. In Section 2, we provide an overview of the Andrews–Curtis conjecture, describing the specific version studied in this work. We then apply classical search algorithms to examples in the Miller–Schupp series in Section 3, where we devise a greedy search algorithm that significantly outperforms the widely used breadth-first search algorithm. In Section 4, we employ reinforcement learning, specifically implementing the Proximal Policy Optimization (PPO) algorithm [Sch+17], and find that while it performs better than breadth-first search, it does not surpass the greedy search algorithm (see Figure 1). Building on these insights, Section 5 presents several ideas for algorithm development, proposing strategies to mitigate the overwhelming complexity faced by an RL agent in challenging problems. In Section 6, we employ topological methods to assess the complexity of presentations. Specifically, we utilize persistent homology to define the *isolation value* of a presentation. This value is infinite for counterexamples to the AC conjecture and serves as a measure of a presentation's resistance to being trivialized. Finally, in Section 7, we examine the linguistic structure of balanced presentations using a decoder-only Transformer model, observing distinct clusters within the embedding space corresponding to presentations solved and unsolved by the greedy search algorithm.

We encourage the reader to explore the accompanying GitHub repository:

https://github.com/shehper/AC-Solver

Acknowledgment. We would like to thank Danil Akhtiamov, Anna Beliakova, Jessica Craven, Michael Douglas, Konstantin Korovin, Alexei Lisitsa, Maksymilian Manko, Ciprian Manolescu, Fabian Ruehle, Josef Urban, and Tony Yue Yu for insightful discussions and comments. We especially want to thank Anna Beliakova for igniting our interest in the Andrews–Curtis conjecture as a framework for exploring problems with long and rare sequences of moves that an RL agent must discover.

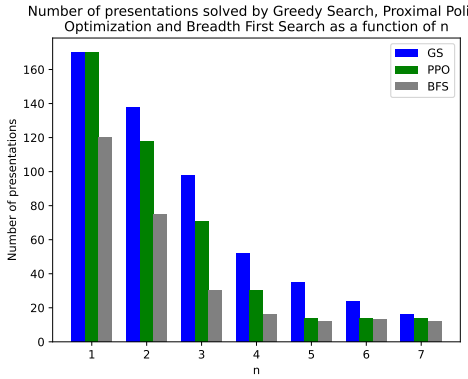
2. Andrews–Curtis conjecture

The Andrews–Curtis conjecture concerns the study of *balanced presentations* of the trivial group, i.e. presentations of the trivial group with an equal number of generators and relators. The conjecture proposes that any balanced presentation of the trivial group

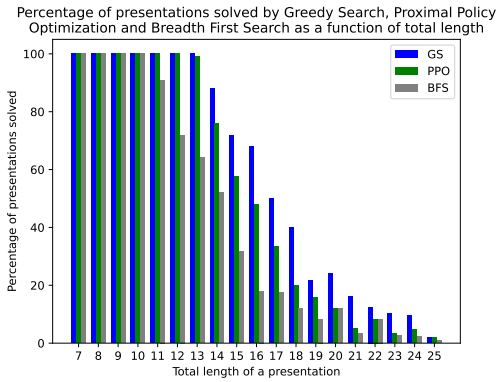
$$\langle x_1, \dots, x_n \mid r_1, \dots, r_n \rangle$$

can be converted to the trivial presentation

$$\langle x_1, \dots, x_n \mid x_1, \dots, x_n \rangle$$



(A) Distribution versus n



(B) Distribution versus length

FIGURE 1. A comparison of three algorithms —breadth-first search, greedy search, and Proximal Policy Optimization (PPO)— that we used to search through the space of balanced presentations. The number of presentations of the Miller–Schupp series, $\mathrm{MS}(n, w)$, solved by an algorithm is given on the vertical axis. We compare the performance as a function of n (above) and the length of the presentation (below). Greedy Search consistently outperforms Breadth-First Search and Proximal Policy Optimization.

through a series of the following operations known as *AC-moves* [AC65]:

(AC1) Substitute some r_i by $r_i r_j$ for $i \neq j$.
(AC2) Replace some r_i by r_i^{-1}.
(AC3) Change some r_i to $g r_i g^{-1}$ where g is a generator or its inverse.

We will refer to the sum of the word lengths of all relators as the *length* of a presentation. Two presentations that can be transformed into each other by a sequence of AC-moves are said to be *AC-equivalent*. A presentation that is AC-equivalent to the trivial presentation is referred to as *AC-trivial*. Despite considerable

efforts, little progress has been made in establishing a proof of the conjecture. At the same time, several families of potential counterexamples have been proposed in the literature.

To investigate a given presentation, one may systematically explore the entire space of possible sequences of AC-moves in search of a sequence that renders the presentation trivial. This space grows exponentially with the length of the sequence. For a presentation with n generators, there are $3n^2$ AC-moves, and the total number of sequences of AC-moves of length k is $(3n^2)^k$. Even for a modest case like $n = 2$ and $k = 20$, the number of possible sequences is on the order of 10^{21}, making a brute-force approach impractical.

Traditional search algorithms such as genetic algorithms [Mia03], and breadth-first search [HR] have been employed to search through this space and achieved success in trivializing balanced presentations with two generators and lengths less than 13. The following presentation of length 13,

$$\langle x, y \mid x^3 = y^4, xyx = yxy \rangle$$

is the shortest presentation, up to AC-equivalence, that eludes all attempts at length reduction. This presentation is a part of an infinite series of potential counterexamples by Akbulut and Kirby [AK85]:

$$\mathrm{AK}(n) = \langle x, y \mid x^n = y^{n+1}, xyx = yxy \rangle, \quad n \geq 2.$$

AK(2) has length 11 and has been established as AC-trivial [Mia03] whereas AK(3) is the aforementioned presentation with length 13.

In over two decades since the first utilization of search algorithms [Mia03; HR], only unsuccessful attempts have been made to trivialize AK(3) with different variants of breadth-first search algorithm using an increasing amount of computational resources [BM06; KS16; PU19]. Notably, [PU19] found that no sequence of AC-moves that allows relator lengths to increase up to 20 trivializes AK(3). This lack of success could be interpreted as suggestive evidence that AK(3) might be a counterexample to the Andrews–Curtis conjecture. However, recent works by Bridson and Lishak have shown that there exist AC-trivializable balanced presentations, for which the number of AC-moves in a trivializing sequence is bounded below by a superexponential function of the length of the presentation [Bri15; Lis17]. Roughly speaking, for these presentations, if the sum of word lengths is k, the number of AC-moves required to trivialize the presentation is at least $\Delta(\lfloor \log_2 k \rfloor)$ where $\Delta \colon \mathbb{N} \to \mathbb{N}$ is defined recursively as $\Delta(0) = 2$ and $\Delta(j) = 2^{\Delta(j-1)}$ for $j \geq 1$. In particular, $\Delta(\lfloor \log_2(13) \rfloor) = 65536$, whereas presentations trivialized by the aforementioned search algorithms have AC sequences of length less than 1000. While AK(3) is itself not a member of the family of examples studied by Bridson and Lishak, their findings challenge the view it as a counterexample. Their work also underscores the necessity of employing search methods that are more efficient than breadth-first search.

In this paper, we will consider a variety of computational tools to better understand the properties of balanced presentations of the trivial group. We will test the efficacy of our approaches on a subset of presentations from the Miller–Schupp series of potential counterexamples [MS99]:

$$\mathrm{MS}(n, w) = \langle x, y \mid x^{-1} y^n x = y^{n+1}, x = w \rangle.$$

Here, $n \geq 1$ is a positive integer and w is a word in x and y with zero exponent sum on x. For $w_\star = y^{-1}x^{-1}yxy$, the presentations $\mathrm{MS}(n, w_\star)$ are AC-equivalent to the presentations from Akbulut–Kirby series [MMS02]. In particular, the presentation

$$\mathrm{MS}(n, w) = \langle x, y \mid x^{-1}y^3x = y^4, x = y^{-1}x^{-1}yxy \rangle.$$

of length 15 is AC-equivalent to AK(3).

We will only consider presentations with $n \leq 7$ and $\mathrm{length}(w) \leq 7$. Our selection criteria aims to strike a balance: we seek a dataset of presentations large enough to allow for meaningful analysis, yet small enough to ensure all computations are feasible within a practical timeframe. We reduce $x^{-1}w$ freely and cyclically, and if two presentations have the same fixed n, but different words w and w' such that letters of $x^{-1}w$ can be cyclically permuted to obtain $x^{-1}w'$, we keep only one of these presentations. After these simplifications, we find 170 choices of presentations for each fixed n, resulting in a dataset of $7 \times 170 = 1190$ presentations for our analysis.

Our implementation of AC-transformations differed from the AC-transformations mentioned above in two ways. First, we considered the following set of operations.

(AC′1) Replace some r_i by $r_i r_j^{\pm 1}$ for $i \neq j$.

(AC′2) Change some r_i to $g r_i g^{-1}$ where g is a generator or its inverse.

For two generators, which is the only case we study in this paper, the group generated by these AC-transformations is isomorphic to the group generated by the original AC-transformations.[3] The reason for this change is due to its effect on performance in greedy search and reinforcement learning algorithms studied in Section 3 and Section 4. Specifically, the length of a presentation provides a useful signal when searching through the space of presentations with these algorithms. An inversion transformation leaves the length invariant providing no signal to the search process and slowing down the performance of the algorithm significantly. For the rest of the paper we will refer to the new transformations (instead of the original AC-transformations) as "AC-transformations" or "AC-moves".

Second, in order to make the search space finite in size, we set a maximum length that each relator is allowed to take. In other words, we mask all AC-moves that lead to presentations with relators exceeding this maximum length. In the search of a sequence of AC-moves that trivialize a presentations of the Miller–Schupp series $\mathrm{MS}(n, w)$, we set this maximum length to be $2 \times \max(2n + 3, \mathrm{length}(w) + 1) + 2$. This specific choice was made to allow for at least one concatenation move followed by a conjugation move in the search process.

3. Classical Search Algorithms

In this section, we compare the effectiveness of breadth-first and greedy search algorithms to AC-trivialize presentations in the Miller–Schupp series. We find that

[3]The difference lies in how the inversion of a relator is handled: we always follow an inversion by a concatenation, while the original AC-moves allow for standalone inversion moves. The original inversion moves may be retrieved from the new generators as follows. For a given presentation $\langle x_1, x_2 \mid r_1, r_2 \rangle$, the sequence of moves: $r_2 \to r_2 r_1$, $r_1 \to r_1 r_2^{-1}$, $r_2 \to r_2 r_1$, and $r_2 \to r_1 r_2 r_1^{-1}$ results in the presentation $\langle x_1, x_2 \mid r_2^{-1}, r_1 \rangle$, which is the same as $r_2 \to r_2^{-1}$ up to swapping the two relators. We also enhanced the notion of trivial presentation(s) to include all presentations of length 2: $\{\langle x_1, x_2 \mid x_i^a, x_j^b \rangle \mid i, j = 1, 2; a, b = \pm 1; i \neq j\}$.

the latter significantly outperforms the former. Additionally, using the greedy search algorithm, we determine that, in the stable case, AK(3) is AC-trivial.

3.1. **Breadth-first search.** We first recall the breadth-first search algorithm. An iterative implementation of this algorithm, adapted to the problem of the Andrews–Curtis conjecture, is provided in Algorithm 1.

We start with an initial state, which is a balanced presentation we wish to AC-trivialize, and place it in a queue. At each iteration, a state is removed from the queue, and its neighbors are added if they have not already been visited. This process continues until the sought-after state, i.e., a trivial balanced presentation, is found or a maximum number of states N is visited. In our experiments, we set $N = 10^6$.

Algorithm 1 Breadth-First Search Algorithm

Input: A balanced presentation π, maximum number of states to visit N
Output: Boolean for whether an AC trivialization is found
Initialize a queue Q and enqueue the starting node π
Mark π as visited
while Number of visited states is less than N **do**
    $u \leftarrow Q$.dequeue() ▷ Remove the front node of Q
    for each neighbor v of u **do**
        if v is a trivial state **then**
            return True ▷ Return True if v is a trivial state
        end if
        if v has not been visited **then**
            Mark v as visited
            Q.enqueue(v) ▷ Add v to the queue
        end if
    end for
end while
return False ▷ Return False if no trivial state is found

3.2. **Greedy search.** The greedy search algorithm, described in Algorithm 2, differs only slightly from the breadth-first search algorithm in implementation. We replace the queue with a priority queue, which stores the states in order determined by a tuple of values (k, l), where k is the length of the presentation and l is the path length between the state and the initial state.

Instead of dequeuing the earliest state, the algorithm dequeues the state with the smallest value of k. If there is more than one state in the priority queue with the same value of k, the state with the smallest value of l is chosen.

3.3. **Comparison.** We find that greedy-search algorithm outperforms the breadth-first search algorithm in the task of AC-trivializing Miller–Schupp presentations Figure 1. Out of the 1190 presentations in the Miller–Schupp series with $n \leq 7$ and length$(w) \leq 7$, greedy search solved 533 while BFS solved only 278 presentations. Each algorithm was constrained to visit a maximum of 1 million nodes. The percentage of presentations solved by these algorithms decreases monotonically as

Algorithm 2 Greedy Search Algorithm

Input: A balanced presentation π of length k, maximum number of states to visit N
Output: Boolean for whether an AC trivialization is found
Initialize a *priority* queue Q ordered by (k, l) and enqueue the starting node π. l is the length of the path connecting π to the current node.
Mark π as visited
while Number of visited states is less than N **do**
    $u \leftarrow Q$.dequeue() ▷ Remove the front node of Q
    for each neighbor v of u **do**
        if v is a trivial state **then**
            return True ▷ Return True if v is a trivial state
        end if
        if v has not been visited **then**
            Mark v as visited
            Q.enqueue(v) ▷ Add v to the queue
        end if
    end for
end while
return False ▷ Return False if no trivial state is found

a function of n. Remarkably, the greedy search was able to solve all presentations with $n = 1$ or length less than 14. There are, however, six presentations of length 14 that greedy search could not solve. We verified that four of these,

$$\langle x, y \mid x^{-1}y^2x = y^3, x = x^{-2}y^{-1}x^2y^{\pm 1}\rangle$$
$$\langle x, y \mid x^{-1}y^3x = y^4, x = y^{\pm 1}x^2y^{\pm 1}\rangle$$

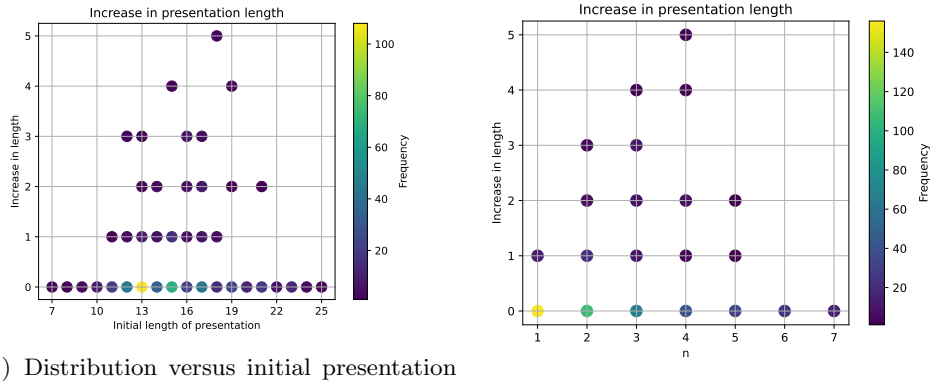
are AC-equivalent to AK(3), while the other two

$$\langle x, y \mid x^{-1}y^2x = y^3, x = yx^2y^{\pm 1}x^{-2}\rangle$$

could be related neither to AK(3) nor to the trivial presentation with any sequence of moves that allowed the length of each relator to increase up to 20.

For presentations solved by greedy search, we plot the maximum amount by which the length of a presentation increased in an AC trivialization path in Figure 2. In most cases, there was no increase in length; and the maximum increase was only 5. At first glance, this seemed surprising to us, given that we allowed the relator lengths to increase by a much larger amount in our search process.[4] However, the hard cutoff set by visiting a maximum of only 1 million nodes ensures that any presentation that needs to be mapped to a much longer presentation before it is trivialized would remain unsolved by the greedy search algorithm. This limitation could be cured either by increasing the number of maximum nodes (at the cost of higher memory use) or by using a different criterion to order nodes in the priority

[4]The length of each relator was allowed to increase up to $2 \times \max(2n + 3, \text{length}(w) + 1) + 2$, which is twice the maximum of the initial lengths of the two relators in a presentation, plus an additional 2. The maximum possible increase in presentation length is twice this number minus the original length. For $n \leq 7$ and $\text{length}(w) \leq 7$, this value lies in the range $[17, 53]$.



(A) Distribution versus initial presentation length.

(B) Distribution versus n.

FIGURE 2. The maximum increase in the length of a presentation relative to its initial length along the AC trivialization path. The increase is plotted as a function of the initial length of the presentation on the left and as a function of n on the right.

queue. It will be useful to explore the latter approach perhaps by looking for a criterion itself using deep learning algorithms.

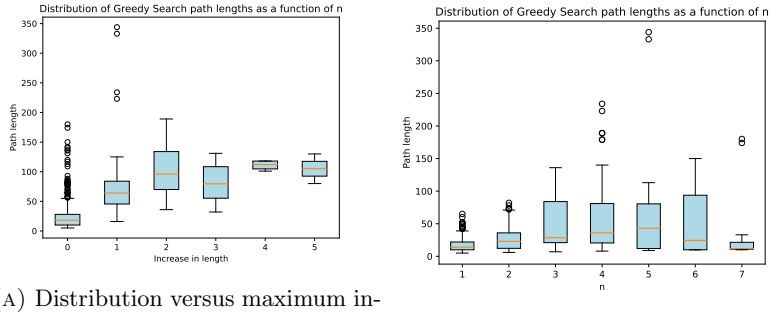
We also plot the lengths of AC sequences discovered by greedy search as functions of n and the maximum increase in the presentation length (Figure 3). Unsurprisingly, path lengths increase proportionally with the increase in the length of the presentation (Figure 3a). The following presentation with $n = 5$ had the longest AC trivialization path,

$$\langle x^{-1}y^5x = y^6 \mid x = yx^2y^{-1} \rangle$$

requiring a sequence of 344 AC-moves. Note that greedy search does not necessarily find the shortest paths of trivialization. We will see in Subsection 4.3 that a Reinforcement Learning algorithm finds shorter trivializing sequences for many examples of the Miller–Schupp series. This again hints at the potential utility of exploring more efficient criteria for ordering nodes in the priority queue.

In the remainder of this paper, we will refer to the presentations from the Miller–Schupp series that were solved and unsolved by the greedy search as "GS-solved" and "GS-unsolved" presentations, respectively. In other words, many of our experiments will be tested on two datasets that consists of Miller–Schupp presentations with $n \leq 7$ and length$(w) \leq 7$: the GS-solved dataset has 533 presentations, whereas GS-unsolved dataset has 657 presentations. The combined dataset that contains all presentations with $n \leq 7$ and length$(w) \leq 7$ has size 1190.

3.4. **Limitations.** While the greedy search algorithm performs better than the breadth-first search, it has some of the same limitations. Namely, it is memory inefficient, and we cannot leverage the parallelizability of modern hardware architectures. It also does not learn a general algorithm that would find an AC trivialization for any given balanced presentation.



(A) Distribution versus maximum increase in presentation length.

(B) Distribution versus n.

FIGURE 3. Distribution of lengths of AC-trivialization paths learned by greedy search as a function of maximum increase in presentation length (left) and n (right).

Reinforcement learning algorithms, particularly policy gradient algorithms, present a promising alternative that avoids these downsides. These algorithms are memory efficient and can be trained in a highly distributed manner, which we will focus on in the next section.

3.5. **Proof of Theorem 1.** As mentioned in Section 1, one nice byproduct of our analysis is that the shortest mysterious AC presentation, namely AK(3), is stably AC-trivial. The goal of this part is to present a proof of this statement.

First, in order to make this part of the paper self-contained, let us remind the reader that the term "stable" (a.k.a. "weak") refers to one of many variants of the Andrews–Curtis conjecture, see e.g. [MMS02; MSZ16; Bag21], where in addition to the usual AC-moves one is allowed to use two more transformations:

(AC4) Include a new generator and a trivial relator, i.e. replace $\langle x_1, \dots, x_n \mid r_1, \dots, r_n \rangle$ by $\langle x_1, \dots, x_n, x_{n+1} \mid r_1, \dots, r_n, x_{n+1} \rangle$.

(AC5) Remove a trivial relator and the corresponding generator, i.e. the inverse of (AC4).

Definition 2. If two balanced presentations of the trivial group are related by a sequence of AC-transformations (AC1) through (AC5), we say that they are *stably AC-equivalent*.

The stable Andrews–Curtis conjecture states that any balanced presentation is stably AC-equivalent to the trivial presentation. To the best of our knowledge, prior to this work, the shortest potential counterexample to the standard Andrews–Curtis conjecture, AK(3), was also a potential counterexample to the stable Andrews–Curtis conjecture. Our proof that AK(3) is stably AC-trivial builds on the following result.

Theorem (Myasnikov, Myasnikov, and Shpilrain, [MMS02])**.** *Using the notation* $[a, b] = aba^{-1}b^{-1}$ *and* $[a, b, c] = [[a, b], c]$*, any presentation of the following form is a presentation of the trivial group:*

$$\langle x, y, z \mid x = z \cdot [y^{-1}, x^{-1}, z],\ y = x \cdot [y^{-1}, x^{-1}, z^{-1}] \cdot [z^{-1}, x],\ w \rangle,$$

where w is a word in x, y, and z whose exponent sum on x, y, and z equals ± 1. Moreover, all such presentations are stably AC-trivial.

These presentations are obtained by applying Reidemeister moves to the knot diagram of the unknot and using the fact that Reidemeister moves applied to a knot diagram give stably AC-equivalent Wirtinger presentations of the knot group, cf. [Wad94].

For $w = x^{-1}yz$, one of the relators eliminates the generator z, resulting in the following length 25 presentation with two generators:

$$\langle x, y \mid x^{-1}y^{-1}xy^{-1}x^{-1}yxy^{-2}xyx^{-1}y,\ y^{-1}x^{-1}y^2x^{-1}y^{-1}xyxy^{-2}x\rangle.$$

We discovered a sequence of AC-transformations (AC1)–(AC5) that relates this presentation to AK(3). This also makes AK(3) the shortest stably AC-trivial presentation that is not yet known to be AC-trivial. It is plausible that by varying w one can show that other presentations of the Akbulut–Kirby series (or the Miller–Schupp series) are also stably AC-trivial. We leave this question for future work.

Specifically, using search algorithms described earlier in this section we placed a cutoff of a maximum of 1 million nodes to visit for each of our search algorithms and allowed the length of each relator to increase up to 15. The greedy search found a path connecting this presentation to AK(3), while breadth-first search could only reduce the presentation's length to 14. We repeated the search process with breadth-first search with a cutoff of 5 million nodes. It failed to reduce the presentation length any further.

The sequence of moves that connects the length-25 presentation to AK(3) can be conveniently expressed in terms of the following 12 transformations:

$$\begin{aligned}
h_1 &= r_2 \to r_2r_1, & h_5 &= r_2 \to x^{-1}r_2x, & h_9 &= r_2 \to xr_2x^{-1},\\
h_2 &= r_1 \to r_1r_2^{-1}, & h_6 &= r_1 \to y^{-1}r_1y, & h_{10} &= r_1 \to yr_1y^{-1},\\
h_3 &= r_2 \to r_2r_1^{-1}, & h_7 &= r_2 \to y^{-1}r_2y, & h_{11} &= r_2 \to yr_2y^{-1},\\
h_4 &= r_1 \to r_1r_2, & h_8 &= r_1 \to xr_1x^{-1}, & h_{12} &= r_1 \to x^{-1}r_1x,
\end{aligned}$$

among which a careful reader can recognize moves (AC′1) and (AC′2) introduced in Section 2. Expressed in terms of the moves h_i, the desired sequence has length 53 and looks as follows:

$$\begin{aligned}
&h_9 \cdot h_7 \cdot h_4 \cdot h_8 \cdot h_{11} \cdot h_5 \cdot h_{11} \cdot h_9 \cdot h_3 \cdot h_{10} \cdot h_{12} \cdot h_7 \cdot h_7 \cdot h_9 \cdot h_{11} \cdot h_5 \cdot h_3 \cdot h_5 \cdot\\
&h_4 \cdot h_3 \cdot h_{12} \cdot h_5 \cdot h_7 \cdot h_7 \cdot h_1 \cdot h_9 \cdot h_{11} \cdot h_8 \cdot h_3 \cdot h_5 \cdot h_{10} \cdot h_2 \cdot h_6 \cdot h_{12} \cdot h_9 \cdot h_7 \cdot\\
&h_5 \cdot h_{11} \cdot h_{10} \cdot h_3 \cdot h_8 \cdot h_{11} \cdot h_9 \cdot h_2 \cdot h_{10} \cdot h_{12} \cdot h_5 \cdot h_7 \cdot h_9 \cdot h_{11} \cdot h_1 \cdot h_9 \cdot h_8.
\end{aligned}$$

This sequence should be read from left to right; first apply h_9, then h_7, and so forth. This follows the standard convention of many programming languages, which iterate over lists from left to right by default. The length of the presentation did not exceed 25 during its path to AK(3). We do not know if this is the shortest path between the two presentations.

4. Reinforcement Learning

This section is organized as follows: in Subsection 4.1, we discuss how the problem underlying the Andrews–Curtis conjecture can be formulated as a Markov Decision

Process. In Subsection 4.2, we discuss details of a specific reinforcement learning algorithm, called the Proximal Policy Optimization algorithm, which we used to find AC trivializations of balanced presentations. Finally, in Subsection 4.3, we discuss the results of our work, comparing the performance of PPO with that of the classical search algorithms studied in the previous section.

4.1. **Markov Decision Process.** A *Markov Decision Process (MDP)* is defined as a 5-tuple (S, A, R, P, ρ). Here, S represents the space of states, while A denotes the set of actions, where each action $a \in A$ is a function mapping from one state to another, i.e., $a\colon S \to S$. The function $R\colon S \times A \times S \to \mathbb{R}$ is the reward function, which assigns a real-valued reward based on the transition from one state to another via a specific action. The transition probability function, denoted by $P\colon S \times A \to \mathcal{P}(S)$, provides the probability distribution over the possible next states given a current state and action. Lastly, ρ represents the initial probability distribution of states, describing the likelihood of the system starting in each state.

The schematic picture of how these objects interact with each other is as follows. We start with a state s_0 sampled from the distribution ρ and take an action a_0. This results in a state s_1 with probability $P(s_1 \mid s_0, a_0)$. The transition gets a "reward" $r_0 = R(s_0, a_0, s_1)$ which quantifies the effectiveness of the action in contributing toward achieving an ultimate goal. From state s_1, we repeat this process, obtaining a trajectory of states

$$\tau = (s_0, a_0, s_1, a_1, \dots)\,.$$

The goal of this process is to maximize the cumulative return,

$$R(\tau) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}).$$

Here, T is the length of the trajectory, known as the "horizon length" and $\gamma \in (0, 1)$ is the "discount factor" that assigns smaller weights to the reward values obtained farther in the future.

For a given problem at hand, we may not *a priori* know the actions $\{a_t\}$ and states $\{s_{t+1}\}$ that maximize the return. Deep reinforcement learning presents a solution to this problem: we train a neural network that learns a map from states to actions with the objective of maximizing the cumulative return. More precisely, we learn a map called the "policy" function $\pi\colon S \to \mathcal{P}(A)$ that assigns to each state a probability distribution over actions. At time step t, an action $a_t \sim \pi(\cdot \mid s_t)$ is sampled that gives the next state s_{t+1}.[5] In the next subsection we discuss the specific algorithm and the objective function that we used in our study.

4.2. **Proximal Policy Optimization.** The goal of our optimization process is to find a policy that maximizes the cumulative return. The most naive way to achieve this goal is through an algorithm known as "vanilla policy gradient." We perform gradient updates guided by the expected return $J(\pi_\theta) \equiv \mathbb{E}_{\tau\sim\pi_\theta} R(\tau)$, where the

[5]In general, we need to specify the probability transition function $P(\cdot \mid s_t, a_t)$ from which the next state would be sampled. In this paper, we take this probability distribution to be a delta function centered at a single state, which we write as $a_t(s_t)$.

expectation is over a set of trajectories consisting of states and actions sampled according to our current policy,

$$\theta_{k+1} = \theta_k + \nabla_\theta J(\pi_\theta).$$

It turns out that this update depends on the gradient of the logarithm of the policy function itself and an "advantage function" $A^\pi(s,a)$ which quantifies the relative benefit of taking action a in state s under policy π [Ach18].[6] Explicitly,

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t,a_t)\right].$$

Thus, vanilla policy gradient algorithm amounts to optimizing the objective function

$$L^{PG} = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t,a_t)\right].$$

Although it has the advantage of being simple, this algorithm has the drawback of making excessively large updates, which can cause the updated policy to deviate significantly from the current policy.

Proximal Policy Optimization (PPO) algorithms seek to make these updates more robust by limiting the extent to which the policy π can change in a single update [Sch+17]. PPO implements this through an objective function that includes a clipped probability ratio between the new policy π_θ and the old policy π_{old}, thus constraining the updates within a predefined range.

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right]$$

Here $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ represents the probability ratio, and ϵ is a small positive constant (commonly set around 0.1 or 0.2) that controls the clipping value. The raw and the clipped ratio ensure that excessively large policy updates are curtailed, making the optimization process more stable.

The advantage function is estimated during the optimization process using the Generalized Advantage Estimation method of [Sch+18]. This method requires an estimation of the on-policy value function (c.f. footnote 6). In PPO, the actor-critic framework is used for this purpose. We train two neural networks in parallel, where the "actor" learns the policy π_θ and the "critic" learns the value function.[7] The value loss L^V is the mean squared error between the values for a state as estimated by the critic network before and after a gradient update. Lastly, PPO adds entropy S of the action-space distribution to the full loss function to prevent premature convergence to a suboptimal policy. The full loss function is as follows,

$$L = \mathbb{E}_t\left[L^{CLIP}(\theta) - c_1 L^V(\phi) + c_2 S(\pi_\theta)(s_t)\right]$$

where c_1 and c_2 are tunable hyperparameters.

[6]Mathematically, the advantage function is the difference between "on-policy action-value function" $Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau) \mid s_0 = s, a_0 = a]$ and the "on-policy value function", $V^\pi(s) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau) \mid s_0 = s]$. These functions give an estimate of the cumulative return given that the current state-action pair is (s,a) in the case of Q^π and the current state is s in the case of V^π.

[7]Sometimes, these neural networks are made to share their parameters, which helps in stability during training. We experimented with shared as well as unshared parameters and observed better performance by keeping the parameters independent.

In policy gradient algorithms such as PPO, we alternate between collecting data through the current policy in the "rollout phase" and updating the policy weights through gradient descent in the "optimization phase". There are various hyperparameters associated to these phases, which we briefly describe now. The rollout phase involves generating multiple trajectories of the underlying Markov Decision Process in parallel using the current policy function. The hyperparameters of this phase include the number of parallel actors N, the rollout length T', i.e. the number of steps each actor takes, the discount factor γ (c.f. Subsection 4.1) and a bootstrapping hyperparameter of Generalized Advantage Estimation λ_{GAE}. The dataset of $N \times T'$ examples collected in this phase is then used to update the policy. In each epoch, the dataset is shuffled and split into mini-batches before performing gradient descent. In addition to the optimizer hyperparameters (such as learning rate schedule), the number of epochs and the size of mini-batches are hyperparameters of the optimization process. Further details of hyperparameters are summarized in Appendix A.

4.3. **Application to the Andrews–Curtis Conjecture.** The set of all balanced presentations of the trivial group with a fixed number of generators and the set of AC-transformations play the roles of sets S and A, respectively. (Recall notations introduced in Subsection 4.1). Once we choose a reward function R and an initial state distribution ρ, we may use the Proximal Policy Optimization algorithm to learn the policy function π. We tested a few different reward functions in our experiments, observing that the following candidate led to the best performance and stability in training.

$$R(s_t, a_t, s_{t+1}) = \begin{cases} -\min(10,\ \text{length}(s_{t+1})) & \text{if length}(s_{t+1}) > 2, \\ 1000 & \text{otherwise.} \end{cases}$$

Here, $\text{length}(s_{t+1})$ is the length of the presentation at timestep $t+1$. The reward function assigns $-\min(10, \text{length}(s_{t+1}))$ to a non-terminal state and 1000 to a terminal state. We found that clipping the reward to -10 led to less variance in gradients of the loss function with respect to weights.

We define the initial state distribution as a distribution over the presentations of the Miller–Schupp series with $n \leq 7$ and $\text{length}(w) \leq 7$. Initially, each presentation was selected exactly once in ascending order by n and $\text{length}(w)$. Following this initial sequence, we maintained an ongoing record of presentations that were either solved or unsolved at any given time. During each rollout phase, a presentation was randomly chosen from the set of solved or unsolved presentations with probabilities of $\frac{1}{4}$ and $\frac{3}{4}$, respectively. This method was designed to allow the policy network to refine its strategies on presentations that were already solved, potentially discovering shorter sequences of AC-moves, while also tackling presentations that remained unsolved.

We also need to choose a horizon length T over which the cumulative return is calculated (c.f. Subsection 4.1). In reinforcement learning, training on tasks with long horizons is usually harder than on those with short horizons. This is because in long horizon tasks it is often more difficult to figure out which actions are responsible for the eventual outcomes, a problem known as the credit assignment problem. There is also more uncertainty in the returns (high variance), which

can slow down convergence and destabilize the learning process. The size of the exploration space also becomes a critical issue. With longer horizons, the agent needs to explore a more diverse set of actions to learn about long-term consequences without getting stuck in suboptimal regions. However, it does not mean that we can not train well-performing agents for long-horizon tasks; rather, it indicates that with longer horizons we may need significantly stronger computational power and extended training periods.[8]

The horizon length T is an important hyperparameter in the context of our problem as any presentation that requires a sequence of AC-moves of length greater than the horizon length will necessarily remain unsolved by PPO. On the other hand, choosing a horizon length that is too long can significantly slow down the training process. Due to the limited amount of hardware available, we mostly experimented with considerably smaller values for T, i.e. $T = 200$ and $T = 400$. With these values, PPO could solve, respectively, 431 and 402 presentations out of the 1190 Miller–Schupp presentations of the initial state distribution.[9] In each case, these presentations formed a subset of the presentations solved by the greedy search.

In the rest of this section, we will describe some observations made with the value $T = 200$.[10] In Figure 1 above, the performance of this experiment is compared with the results of the greedy search and the breadth-first search. While PPO consistently outperformed BFS for all values of n and for all lengths of presentations, it consistently underperformed compared to the greedy search.[11] In Figure 4a, we plot the distribution of path lengths discovered by greedy search in the two cases of presentations that could / could not be solved by PPO. It is clear that the presentations PPO solved had, in general, smaller path lengths. In particular, all of these had greedy search path lengths less than 200.

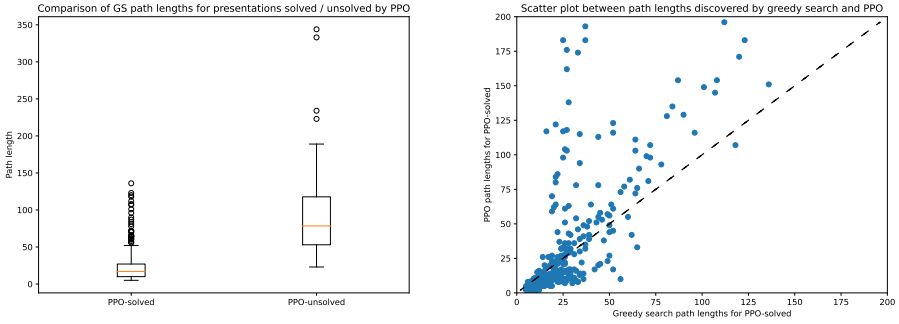
In Figure 4b, we give a scatterplot of the path lengths discovered by PPO and greedy search for all of the presentations solved by PPO. We note that in many cases, PPO found shorter paths compared to the greedy search. This is expected as PPO learns to improve its strategy during the training, discovering shorter paths for presentations it may have already solved. The scatterplot shows the shortest paths discovered by PPO for each presentation. We also note that in many cases, PPO found longer paths than greedy search. This shows that our specific run exhibits a suboptimal policy. It could perhaps be improved by performing more hyperparameter tuning on the training process.

[8]Another option for improving performance in long-horizon tasks is to provide good intermediate rewards; unfortunately, this is rather hard in the present context of the AC conjecture.

[9]We also explored the value $T = 2000$. However, we found it much slower to train due to the reasons described above. We could only solve 219 presentations of the initial state distribution in this case. This training run had not converged, and we expect that with more computational power and extended training periods, it will be worthwhile to experiment with larger values of T, perhaps helping us solve even more presentations than greedy search.

[10]The complete list of hyperparameters for this experiment is summarized in Appendix A.

[11]The results in this section were obtained using relatively small neural networks for both the actor and the critic networks, each consisting of two layers with 512 hidden units. It is natural to assume that increasing the size of these networks could enable a PPO agent to outperform the greedy search. However, such experiments were not performed due to computational limitations. Future work could explore this direction to assess the impact of larger network architectures.



(A) (B)

FIGURE 4. A comparison of path lengths discovered by the greedy search and a PPO agent. The left panel shows the distribution of lengths of AC trivialization paths discovered by the greedy search in the cases solved / unsolved by PPO. The right panel shows the scatter plot of path lengths discovered by PPO vs path lengths discovered by the greedy search.

5. The Cure: New Algorithms

In previous sections we explained from a variety of different perspectives that the Andrews–Curtis conjecture is a good example of a mathematical problem where the length of a solution can be much greater than the length of the initial presentation, in some cases with purely analytical lower bounds that are hyperexponential in the size of the input. In particular, we saw that small increases in presentation length under 20 quickly lead to solution lengths in the range of hundreds and higher, quickly exceeding the number of moves in the longest game of chess.

If solving a mathematical problem required finding a path of length L, say with $L = 10^6$, an RL agent would be pretty much out of luck under circumstances of a typical hard search problem, where the number of successful paths is exponentially suppressed by L. The good news is that in mathematics —and in many other domains— such hard search problems never come in isolation. Rather, there is a distribution of problems such that generic instances are "easy" and a small fraction is "hard." Learning this distribution for smaller values of L contains the crucial information for solving new cases at the next increment of L.

5.1. **Supermoves.** In automated reasoning or search problems where the minimal length solution has a theoretical lower bound that by far exceeds computational capabilities, it is clear that direct approach with fixed size steps is not going to succeed, unless the problem is easy and a large fraction of long paths meets the desired criteria. In order to reach extraordinary path lengths, one must allow progressively longer sequences of elementary moves to be added to the action space. Although this general strategy seems unavoidable in problems like the AC conjecture, it leads to many practical questions. For example, what should be the selection

criteria for such "supermoves"? And, how often should they be added to the action space?

In the context of the AC conjecture, a good example of such supermoves are the "elementary M-transformations" [BM93; Bur+99]. These transformations trivialize AK(2) in just two steps, even though this presentation is known to admit the shortest AC trivialization path of length 14. A downside of elementary M-transformations, though, is that they are infinite in number, which complicates their application in classical search techniques.

In our study, we explored the idea of identifying AC supermoves by selecting some frequently occurring subsequences of AC-moves in the paths discovered by Proximal Policy Optimization (PPO). By extending the action space A of the Markov Decision Process (MDP) with these subsequences and checking whether this enhanced action space helps our agent discover shorter paths of trivialization, we learned a few useful lessons:

- First, it helps to augment the action space with subsequences of different kind that include frequently occurring compositions of elementary moves as well as very rare ones.
- Also, in the early stage it helps to introduce several supermoves at once.
- And, at later stages it helps to allow removing actions from the action space, not only adding them.

Not following these empirical rules, e.g. introducing too few supermoves initially or too many over the entire length of the training process, leads to considerable reduction in performance of the RL agent. Even in the most optimal regimes that we were able to find, the improvement of the performance due to supermoves was rather modest, leading us to explore other alternatives.

5.2. **The anatomy of success.** While supermoves clearly need to be a part of the solution in hard problems like the AC conjecture, much of the success depends on the criteria for selecting them. Here, we advocate for a dynamic approach where the network itself learns the criteria for selecting supermoves, in addition to the best ways to implement them. One realization of this approach could be a multi-agent model, where one network is learning to play the game and the other is learning the rules for changing the action space (adding and removing supermoves). We hope that future iterations of this strategy can lead to AI systems that can 'learn how to learn' dynamically by making both algorithmic and architectural changes through collecting the information about hard instances.[12]

Specifically, suppose N is one of the characteristics of either the algorithm or the architecture that has non-trivial impact on performance. In practice, there can be several such parameters, but for simplicity we explain the idea as if there is only

[12]Here, by self-improving AI systems we mean algorithms that have the ability to "interpolate" between off-the-shelf algorithms such as A2C and TRPO, as well as a myriad of custom algorithms that do not even have a name. Clearly, this level of technology is not presently available, and one of the key points of this section is that developing such systems should be based on the hardest instances the agent encounters.

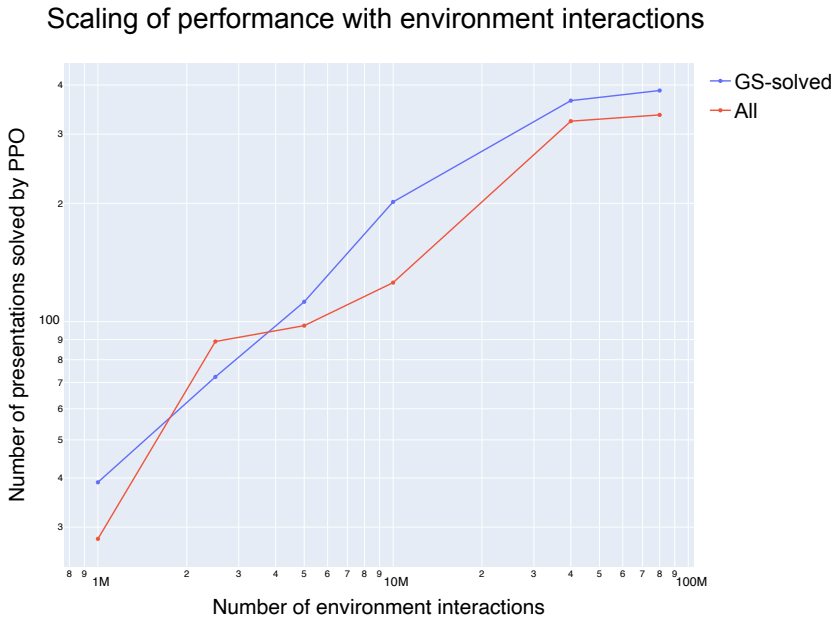


FIGURE 5. Number of AC presentations solved by our PPO agent as a function of the number of training steps. Here, *GS-solved* refers to a subset of the Miller–Schupp dataset of Section 3 that was solved by the greedy search algorithm.

one.[13] Then, from the practical standpoint, a natural notion of hardness is such that *hard instances* are defined to be those which the model can solve at the current setting of N and not with the lower value of the resource N. In addition, in search problems we include the length of the path in the notion of hardness, i.e. select a subset of the instances that the model could solve through especially long paths. Note, by the very nature of the search problem we are interested in, there can not be too many such hard instances at each step of increasing N, for otherwise the problem would be easy, not hard. Collecting the information about the hardest instances at each increment in N can be used to select supermoves, e.g. as subsequences of the sequences of moves that solve the hard instances. Algorithm 3 provides one particular realization of this idea.

In the context of the AC conjecture, examples of the metric N can be the horizon length or the number of interactions with the environment. As Figure 5 illustrates, increasing the number of environment interactions leads to a larger number of non-trivial presentations from the Miller–Schupp series being solved (i.e. AC-trivialized) by our RL agent. Moreover, the length of the AC trivialization path also grows for some of the solutions (but not all). Therefore, in order to implement the program

[13]Analyzing a multi-dimensional landscape is generally a good idea, as it can yield better performance improvements, but the price to pay is that changes of different characteristics become correlated, as illustrated in Figure 6.

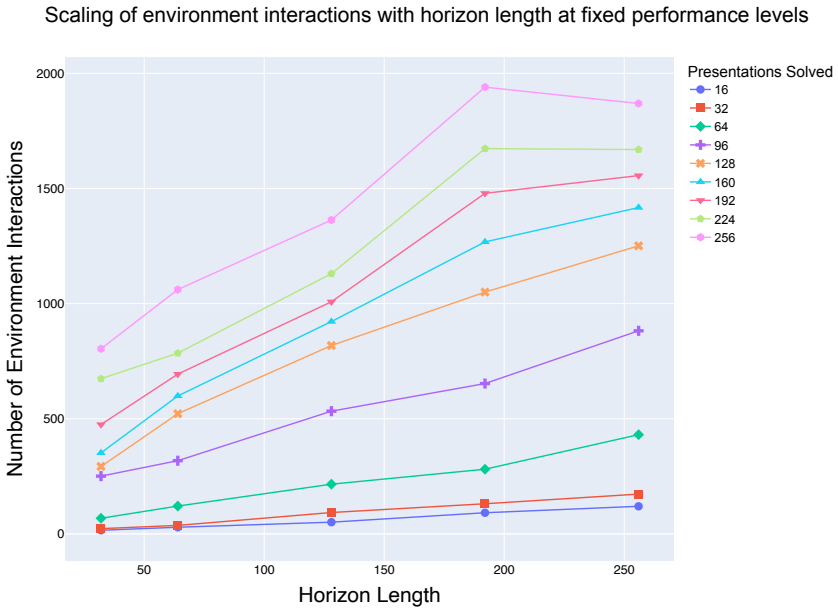
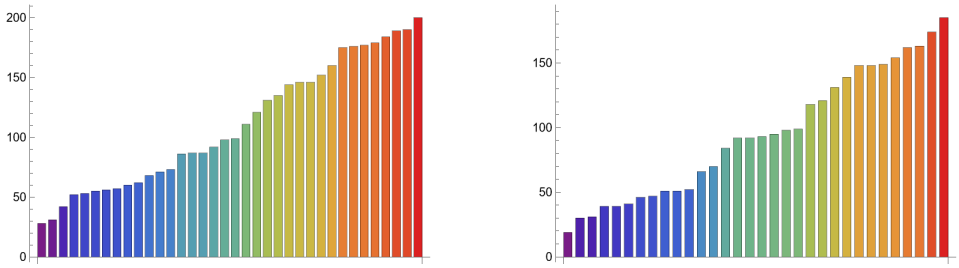


FIGURE 6. To maintain consistent performance, increasing the horizon length requires a roughly linear increase in the number of training steps (environment interactions).

outlined above in the specific context of the AC conjecture, we can focus on the longest AC trivialization paths that the model is able to find at each value of N.



(A) all (B) GS-solved

FIGURE 7. Path length distributions for AC presentations solved by the RL agent at $N = 8 \times 10^7$ from *all* and *GS-solved* datasets are nearly identical. In both cases, hard instances are shown in red.

Collecting this information in the process of training can be used to introduce (and remove) supermoves to the action space in a dynamic fashion. There can be many different implementations of this idea that we plan to explore more fully elsewhere. For example, one can consider selecting all or subset of the longest AC

Algorithm 3 Adaptive AI Model Training and Path Discovery

Input:
Family of AI models $\pi(N)$ with common state space S and action space A_0
Initial setting N_0 and ordered range $\{N_1, N_2, \ldots, N_{\max}\}$
Number of epochs for training
Validation set $V \subset S$
Distinguished state $s_0 \in S$
Positive integer n
Output:
For each setting N_i: Set of pairs $\{v, P\}$ where $v \in V$ and P connects v to s_0
Initialize $A(N_1) \leftarrow A_0$
for each N_i in $\{N_1, N_2, \ldots, N_{\max}\}$ **do**
    Train model $\pi(N_i)$ on S for the given number of epochs
    Evaluate $\pi(N_i)$ on V to discover paths connecting V to s_0 using $A(N_i)$
    $V(N_i) \leftarrow \{v \in V \mid v$ can be connected to s_0 using $A(N_i)$, but not by any $\pi(N_j)$ with $j < i\}$
    $W(N_i) \leftarrow \{v \in V(N_i) \mid$ the longest path connecting v to s_0 using $A_0\}$
    if $i \geq n$ **then**
        Compare $W(N_{i-n+1})$ to $W(N_i)$
        Adjust $A(N_{i+1})$ based on the comparison
    else
        $A(N_{i+1}) \leftarrow A(N_i)$
    end if
end for

trivialization paths that the model finds at each N. Out of those, in each case, one can consider selecting the entire trivialization path or a subset, randomly or non-randomly. Alternatively, one can compare the longest trivialization paths at several (consecutive) values of N and choose subsequences of moves that are shared by several long trivialization paths at different N.

For example, if N denotes the number of interactions with the environment, we did a few preliminary experiments with the dataset of Section 4 and several different seed values. To illustrate the effects of stochasticity, let us consider $N = 8 \times 10^7$. The agents with five different seed values were able to solve 354, 337, 330, 328, and 323 presentations, respectively. And their average, 334.4, is shown in Figure 5. Many of these AC presentations can be solved at the earlier stage, with $N = 4 \times 10^7$ or less. If in the definition of *hard instances* we require that they are solved by *all* five agents, there are only 5 presentations total. On the other hand, if we require that they are solved by *any* of the 5 agents, the number goes up to 36.

Moreover, not surprisingly, the 5 presentations solved by all 5 agents have considerably shorter path lengths (47, 31, 22, 14, and 13) compared to path lengths of the 36 presentations illustrated on the left panel of Figure 7 that go up to 200. Both 5 presentations solved by all agents and 36 presentations solved by at least one of the agents provide viable options for defining hard instances and, in turn, selecting supermoves. However, they lead to qualitatively different results. For

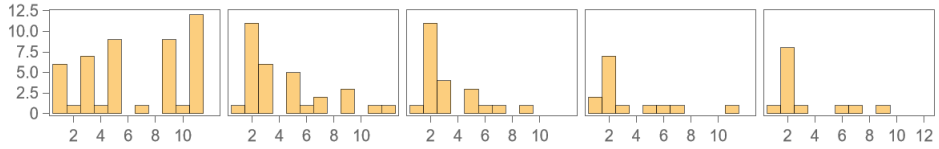


FIGURE 8. Types of AC-moves that appear in trivialization paths of 5 presentations solved by all 5 agents at $N = 8 \times 10^7$. The move #2 occurs disproportionately more frequently. There are 12 different types of basic AC-moves described in Section 3.

example, all 5 presentations solved by all 5 agents are solved at a smaller value of N when required to be solved by only one of the agents. More importantly, they have very different anatomy, illustrated in Figure 8 and in Figure 9, respectively. By examining the longest trivialization paths of the 36 presentations solved by at least one agent at $N = 8 \times 10^7$, we often see long strings of moves #5 and #11, interlaced with moves #3, #7, and #9. These are our top candidates for the supermoves to be added at $N = 8 \times 10^7$. Note that moves #4 and #8 are least common in the examples presented in both Figure 8 and Figure 9.

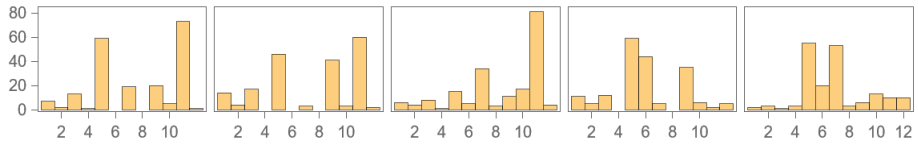


FIGURE 9. Types of AC-moves in the 5 longest trivialization paths (of length 200, 190, 189, 184, and 179) found by at least one agent at $N = 8 \times 10^7$. The most frequent moves are #5, #7, #9, and #11. There are 12 different types of basic AC-moves described in Section 3.

As in other parts of this paper, we performed the analysis on two datasets of sizes 1190 and 533 that, respectively, contain all members of the Miller–Schupp family with $n \le 7 \wedge \text{length}(w) \le 7$ and only those solved by the greedy search. The results are qualitatively similar, as we already saw in Figure 7 that illustrates length distributions of the successful AC paths in the two cases. Similarly, a closer look at the anatomy of the successful paths —successful for the RL agent— reveals no qualitative differences between the two datasets and, importantly, consistency of our notion of *hardness* based on the path length. The largest level of stochasticity that one may expect perhaps can be illustrated by an example of the presentation

$$\langle x, y \mid x^{-1}y^2xy^{-3} = 1\,,\ x^{-2}y^{-1}xy^{-4} = 1 \rangle$$

that an RL agent was able to solve at $N = 4 \times 10^7$ with 62 moves in one case and at $N = 8 \times 10^7$ with 200 moves in the other case. Despite considerable variance, in both cases successful AC paths are dominated by the move #11, interlaced with moves

#3, #5, #7, and #9 according to patterns described above (cf. Figure 10). This can serve as evidence for robustness of the supermove selection process proposed here, and provides further support to Algorithm 3 and its variations.

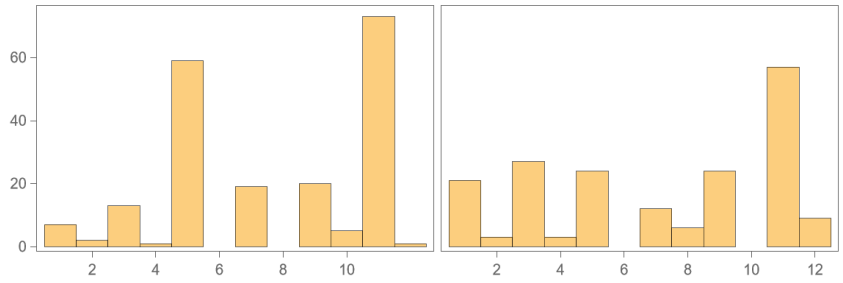


FIGURE 10. An example illustrating anatomy of successful AC paths found by an RL agent in different runs working on different datasets, both of which contain the same hard instance. Even though the total number of moves can vary, e.g. in this example we had to rescale one of the distributions by a factor of 3, curiously, the proportion of moves and combinations used are quite similar. In both cases, RL agents used the same opening move.

In the following sections we explore the anatomy of successful AC paths further using the tools of unsupervised learning and topological data analysis. These tools offer a complementary perspective on hard-to-find AC paths and ways to find them.

6. Isolated components and neighborhood sizes

6.1. **Isolated components.** In Subsection 3.2, we explored a greedy approach to finding AC-trivializations of a presentation π. Specifically, the goal was to construct a sequence of presentations $(\pi_0, \dots, \pi)$, where π_0 is the trivial presentation, such that each consecutive pair in the sequence is related by an AC-move. Furthermore, at each step k, the presentation π_k was chosen to have the shortest possible length among all presentations connected to π_{k+1} via an AC-move. In general, the length of a presentation in an AC-trivialization tends to exceed the length of the original presentation being trivialized. The minimum increase in length across all possible AC-trivializations serves as an invariant of the presentation. We will explore this invariant using concepts from persistent homology.

6.1.1. *Formalization.* A *based graph* is a pair (Γ, v_0) consisting of a graph Γ and a preferred vertex v_0 in it. We will often drop v_0 from the notation. A *based subgraph* Γ_n of Γ, written $\Gamma_n \leq \Gamma$, is a subgraph Γ_n of Γ with the same preferred vertex. We say that Γ_n is *full* in Γ if for any two vertices in Γ_n joined by an edge in Γ, the edge is also in Γ_n. A *filtration* of a based graph Γ is a collection

$$\Gamma_0 \leq \Gamma_1 \leq \Gamma_2 \leq \cdots$$

of based subgraphs of Γ for which each vertex and edge of Γ is in Γ_n for some n. We refer to Γ as a *filtered based graph*. If each Γ_n is full in Γ we refer to the filtration as

full and notice that full filtrations are equivalent to $\mathbb{N}$ valued functions from the set of vertices of Γ sending v_0 to 0.

Let $\Gamma^{\mathrm{AC(k)}}$ be the graph whose vertices are k-balanced presentations, based at the trivial presentation, having an edge between two vertices if there is an AC-move between them. Additionally, $\Gamma^{\mathrm{AC(k)}}$ is equipped with a full filtration obtained from the function sending a vertex to the length of its presentation minus k.

Given a filtered based graph (Γ, v_0). The *filtration value* $\mathrm{Filt}(v)$ of a vertex v is the smallest $n \in \mathbb{N}$ such that v is a vertex in Γ_n. Similarly, its *connectivity value* $\mathrm{Conn}(v)$ is the smallest $n \in \mathbb{N}$ such that v and v_0 can be joined by a path in Γ_n or is set to ∞ if such path does not exist in Γ.

The *isolation value* of a vertex v in a filtered based graph is defined as

$$\mathrm{Isol}(v) = \mathrm{Conn}(v) - \mathrm{Filt}(v),$$

a number in $\mathbb{N} \cup \{\infty\}$. A vertex is said to be *isolated* if its isolation value is positive.

We introduce an equivalence relation on isolated vertices. Two belong to the same *isolated component* if they have the same connectivity value, say n, and they can be joined by a path in Γ_{n-1}. The *isolation value* of a component is the maximum of the isolation values of its elements.

We can interpret these invariants using the framework of topological data analysis, see for example [CV22]. Specifically, the set of isolated components of a based filtered graph Γ corresponds to the multiset of bars in the barcode of its reduced persistent 0-homology. Additionally, the isolation value of an isolated component corresponds to the length of its associated bar.

6.1.2. *Experimental results.* Let Γ^{ℓ} be the full subgraph of $\Gamma^{\mathrm{AC(2)}}$, with the induced filtration, consisting of all presentations with a connectivity value less than or equal to ℓ. Explicitly, Γ^{ℓ} includes all vertices that can be connected to the trivial vertex via paths containing only presentations of length at most ℓ.

We will denote by $v(\ell)$ and $e(\ell)$ the number of vertices and edges of Γ^{ℓ}. Let us denote by $ic(\ell)_k$ the number of isolated components with isolation value k. Figure 11 summarize our results for the classic AC-moves whereas Figure 12 does so for their prime version.[14]

6.2. **Neighborhoods.** Let us return to our data set of 1190 presentations in the Miller–Schupp series for $n \leq 7$ and $\mathrm{length}(w) \leq 7$. Using the methods described in Subsection 4.2, we trained a PPO agent that successfully solved 417 of these presentations. We will refer to the set of these 417 presentations as PPO-solved and the remaining 773 presentations as PPO-unsolved. Our goal is to analyze the relationship between these labels and the sizes of their respective AC neighborhoods. A presentation is considered to be in the k-step neighborhood of another if they can be connected by applying at most k AC-moves.

[14]For this task we used `giotto-TDA` version 5.1 [Tau+21]. Specifically, its binding of the `SimplexTree` data structure introduced in [BM14] and implemented in `GUDHI` [Mar+14].

ℓ	$v(\ell)$	$e(\ell)$	$ic(\ell,1)$	$ic(\ell,2)$	$ic(\ell,3)$
3	36	72	0	0	0
4	100	248	0	0	0
5	388	1072	0	0	0
6	884	2376	0	0	0
7	3892	10775	0	0	0
8	9172	25675	0	0	0
9	37428	106513	0	0	0
10	84996	239733	0	0	0
11	350356	1002439	4	0	0
12	791140	2251375	16	0	0
13	3238052	9321629	72	4	0
14	7199908	20573343	144	4	0
15	29243812	84391763	508	52	8
16	64623652	185162236	1034	88	20

FIGURE 11. Classical AC-moves

ℓ	$v(\ell)$	$e(\ell)$	$ic(\ell,1)$	$ic(\ell,2)$	$ic(\ell,3)$
3	36	40	3	0	0
4	100	152	3	0	0
5	388	712	3	0	0
6	884	1528	3	0	0
7	3892	6984	3	0	0
8	9172	16728	3	0	0
9	37428	69752	3	0	0
10	84996	155752	3	0	0
11	350356	655928	19	0	0
12	791140	1467080	67	0	0
13	3238052	6107112	243	16	0
14	7199908	13414744	483	16	0
15	29243812	55306744	1819	136	32
16	64623652	120824232	3923	208	80

FIGURE 12. Prime AC-moves

6.2.1. *Experimental results.* There are 131 distinct neighborhood sizes in our data. Their basic statistics are

Min	**Max**	**Mean**	**Median**
72,964	89,872	89,532	89,859

A more detailed description of the frequency of values is presented in Figure 13.

The largest neighborhood size accounts for nearly a third of all considered presentations. However, it represents only 2.4% of PPO-solved presentations, while

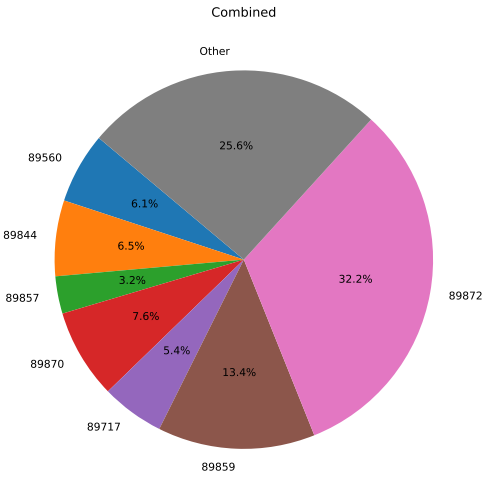


FIGURE 13. Sizes of the 5-step neighborhood of all considered presentations in the Miller–Schupp series. We group neighborhood sizes whose representation is below 2.5%.

constituting almost half (48.3%) of the PPO-unsolved presentations. For more details, please refer to Figure 14.

In contrast, using BFS, these proportions are 7.1% and 52.5%, respectively.

Another notable feature visible in Figure 14 is that just three neighborhood sizes account for over three-quarters of all PPO-unsolved presentations. When considering six neighborhood sizes, this proportion rises to 96.9%. In fact, only twelve neighborhood sizes are present among PPO-unsolved presentations, whereas all 131 sizes appear among PPO-solved presentations. The most common neighborhood size for PPO-solved presentations is 89,560, representing only 17.3% of them. Moreover, 54.2% of all PPO-solved presentations have a neighborhood size shared by less than 2.5% of other PPO-solved presentations.

As we observed, having a maximal neighborhood size provides significant insight into whether a presentation is labeled as PPO-solved or PPO-unsolved. Additionally, the minimum neighborhood size among PPO-unsolved presentations—89,573—is also quite telling, as 54% of PPO-solved presentations have neighborhood sizes smaller than this value. This percentage can be further improved by considering that the neighborhood sizes of PPO-unsolved presentations are concentrated within three specific bands. Please refer to Figure 15 for more details. We find that 64.3% of PPO-solved presentations fall outside the three bands $[89,575, 89,575]$, $[89,715, 89,831]$, and $[89,844, 89,872]$, which together contain over 99% of PPO-unsolved presentations. By narrowing the last band to $[89,859, 89,872]$, these three bands now encompass the neighborhood sizes of over 90% of PPO-unsolved presentations, while their complement includes 77.2% of PPO-solved presentations.

One might expect that enhancing the discriminatory power of n-neighborhoods could be achieved by incorporating features beyond their size. We explored two additional types of features, but surprisingly, they only marginally improved the

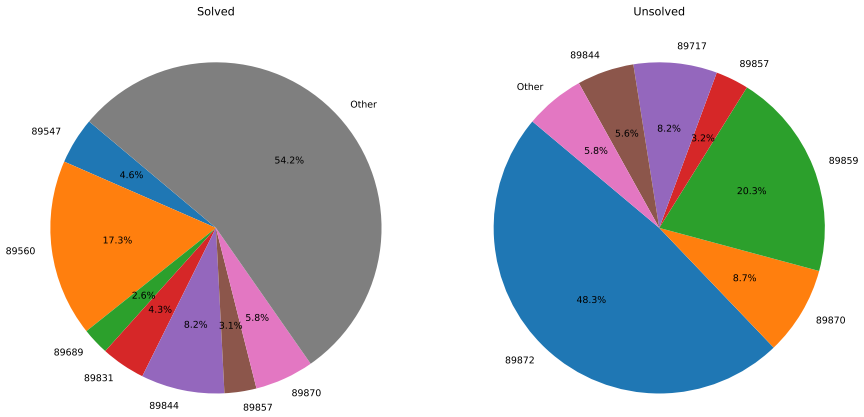


FIGURE 14. Pie charts for the neighborhood size of PPO-solved and PPO-unsolved presentations. We grouped sizes with representation below 2.5%.

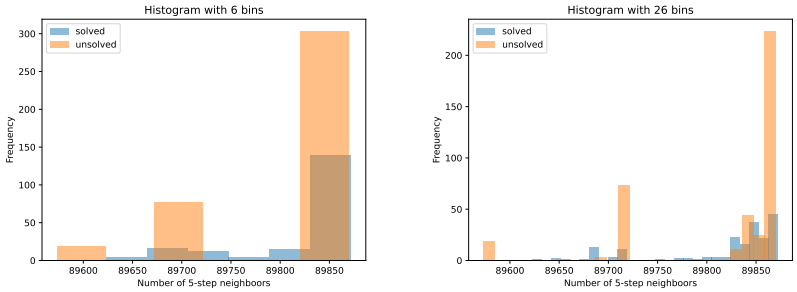


FIGURE 15. Histograms with 6 and 26 bins respectively of the neighborhood sizes of the 417 PPO-solved and 773 PPO-unsolved presentations.

accuracy of PPO-solved/unsolved predictions. The first type was based on node centrality, while the second focused on spectral features of the neighborhood graphs. The latter was particularly intriguing, given the emphasis on Markov processes and the well-known relationship between random walks on graphs and the graph Laplacian.

7. Language Modeling

In this section, we discuss a model for the "language" of balanced presentations. Each presentation with two relators is a sequence made of six letters, also known as "tokens" in the nomenclature of Natural Language Processing, i.e. x, y, x^{-1}, and y^{-1}, and two "stop tokens": one that separates two relators of a presentation and another that marks the end of a presentation. Given this vocabulary V of six tokens, we can ask what is the probability $p(t_1, \ldots, t_N)$ for $t_i \in V$ of the occurrence

of a specific presentation in the space of all balanced presentations. Using the chain rule of probability theory,

$$p(t_1 \cdots t_N) = \prod_{i=1}^{N} p(t_i \mid t_1 \cdots t_{i-1})$$

Here $p(t_N \mid t_1 \cdots t_{N-1})$, often called the N-gram probability distribution, is the probability of a token t_N following a sequence of tokens $t_1 \cdots t_{N-1}$. To model the language of balanced presentations, we can alternatively estimate the N-gram probability distributions for all N.

Over the last few years, Transformer models have shown great success in modeling human-understandable languages to the extent that these models can create text almost indistinguishable from that of a human expert. Specifically, the architecture used for modeling language is the auto-regressive "decoder-only" Transformer, which we review in detail in Subsection 7.1. In Subsection 7.2, we discuss the method with which we generate the dataset required for training the model. Finally, in Subsection 7.3, we share details of some insights we learned from this process.

7.1. **Transformers: a review.** Here, we give a short review of the architecture of a decoder-only transformer. For more details, see [Vas+23; Elh+21; Dou23].

Given an input sequence $t_1, t_2, \ldots, t_N$, a decoder-only transformer predicts the probability distribution $p(t \mid t_1, t_2, \ldots, t_N)$ over the set V of tokens of size n_{vocab}. The probability is computed by applying the softmax function to the logits $T(t)$, which are estimated by applying the following sequence of operations.[15] First, assign to each token in the vocabulary a distinct label in the range $1, 2, \ldots, n_{\text{vocab}}$; re-writing the original sequence as a sequence of integers. We will label these integers also as t_i. Next, write the sequence in terms of "one-hot encoded vectors", i.e. a matrix $t \in \mathbb{R}^{N \times n_{\text{vocab}}}$ such that

$$t_{ij} = \delta_{it_i}$$

and embed the sequence in a d_{model}-dimensional vector space,[16]

$$x_0 = (W_P \otimes \mathbb{1} + \mathbb{1} \otimes W_E)t$$

Here, $W_P \in \mathbb{R}^{d_{\text{model}} \times N}$ and $W_E \in \mathbb{R}^{d_{\text{model}} \times n_{\text{vocab}}}$ are matrices of learnable parameters, known as the "positional embedding" and "token embedding" matrices.

An L-layer transformer alternates between applying a "multi-head attention layer" ($\sum_{h \in H} h$) and an "MLP-layer" (m) L times. For $i = 0, \ldots, L-1$,

$$x_{2i+1} = x_{2i} + \sum_{h \in H} h(\mathrm{LN}(x_{2i})),$$
$$x_{2i+2} = x_{2i+1} + m(\mathrm{LN}(x_{2i+1})).$$

---

[15] The softmax function, $\mathrm{Softmax}\colon \mathbb{R}^n \to (0,1)^n$, is defined as $\mathrm{Softmax}(x)_i = e^{x_i} / \sum_{j=1}^{n} e^{x_j}$.

[16] Here, t and all x_j are two-dimensional tensors. Hence, it is appropriate to apply tensors of linear transformations to them. Often in a transformer architecture, these operations are of the form $\mathbb{1} \otimes \cdots$; in these cases, we drop the identity transformation and simply write the operation as $\cdots$. For example, $\mathbb{1} \otimes W_U$, $\mathbb{1} \otimes W_I^m$, $\mathbb{1} \otimes W_O^m$, etc. In this case, we will sometimes write W_U, W_I^m, W_O^m respectively, assuming it is clear from the context and the dimensionality of these matrices that they are tensored with identity transformations.

Each x_j is an element of $\mathbb{R}^{\mathbb{N}\times d_{\text{model}}}$, with the interpretation that its i-th row is the embedding of the sequence $t_1,\dots,t_i$ in the embedding space $\mathbb{R}^{d_{\text{model}}}$ as learned by the preceeding $j+1$ operations. Finally, one applies an "unembedding layer", $W_U \in \mathbb{R}^{n_{\text{vocab}}\times d_{\text{model}}}$, to convert the output of the final layer to an n_{vocab}-dimensional vector of logits that estimate the sought-after probability distribution.

$$T(t) = W_U x_{2L-1},$$
$$p(t) = \text{Softmax}(T(t)).$$

The functions LN, m and h are defined as follows. LN is the LayerNorm operation that normalizes the input of each layer to make the optimization process more stable ([BKH16]):

$$\text{LN}(x) = (\mathbb{1}\otimes \text{diag}(\gamma))\,\frac{(x-\overline{x})}{\sqrt{\text{var}(x)}} + \mathbb{1}\otimes\beta\,.$$

Here, $\overline{x}$ and $\text{var}(x)$ are mean and variance of each row of x, and $\gamma,\beta\in\mathbb{R}^{d_{\text{model}}}$ are learnable parameters. The MLP-layer m is a non-linear operation,

$$m(x) = W_O^m\ \max(W_I^m x, 0)$$

with learnable parameters $W_I^m \in \mathbb{R}^{d_{\text{MLP}}\times d_{\text{model}}}$, $W_O^m \in \mathbb{R}^{d_{\text{model}}\times d_{\text{MLP}}}$. It is standard to set $d_{\text{MLP}} = 4d_{\text{model}}$.

Finally, the multi-headed attention-layer $\sum_{h\in H} h$ is a sum of n_{heads} "attention-head" operations h, where

$$h(x) = (A^h(x)\otimes W_O^h W_V^h)x.$$

Here, $W_V^h \in \mathbb{R}^{d_{\text{head}}\times d_{\text{model}}}$, $W_O^h \in \mathbb{R}^{d_{\text{model}}\times d_{\text{head}}}$ are matrices of learnable parameters; d_{head} is the "attention-head dimension" that satisfies $d_{\text{head}}\times n_{\text{head}} = d_{\text{model}}$; and the attention matrix A^h is computed with the help of learnable matrices $W_Q^h, W_K^h \in \mathbb{R}^{d_{\text{head}}\times d_{\text{model}}}$,

$$A^h(x) = \text{Softmax}^\star\left(\frac{x^T(W_Q^h)^T W_K^h x}{\sqrt{d_{\text{head}}}}\right).$$

The attention-head is an $N\times N$ matrix, with the interpretation that $A^h(x)_{ij}$ is the "attention" paid to the token t_j in estimating $p(t_{i+1}\mid t_1,\dots,t_i)$. Softmax* is a variant of the Softmax function suitable for auto-regressive tasks: it sets the upper triangular part of its input to zeros before applying the Softmax operation. That is, future tokens, t_k for $k>i$, play no role in the prediction of $p(t_{i+1}\mid t_1,\dots,t_i)$.

We train the transformer model by minimizing the cross-entropy loss between the distributions of predicted and correct labels for the next tokens in a sequence. The parallelism offered by the processing of all tokens in a sequence at once is extremely beneficial for efficient training of the model for the language modeling task.

In practice, the embedding matrix W_E and the unembedding matrix W_U are often "tied" together, i.e. $W_E = W_U^T$ [PW17; IKS17]. The rows of $W_E = W_U^T$ are interpreted as the embeddings of words/sentences, to which one may apply the usual operations of a vector space [Ben+03; MYZ13]. For example, the cosine of the angle between two embedding vectors, also known as the "cosine similarity", is often used to measure the similarity between two texts. Two semantically similar

texts have higher cosine similarity between them, while semantically different texts correspond to (almost) orthogonal vectors in the embedding space.

7.2. **Training and Evaluation Datasets.** We now discuss the training and validation datasets used to train and evaluate our Transformer model. As our main interest in this paper has been in the presentations of the Miller–Schupp series, we generated a dataset of balanced presentations that are AC-equivalent to the Miller–Schupp presentations. Specifically, we apply sequences of AC-moves to the 1190 presentations with $n \leq 7$ and $\text{length}(w) \leq 7$ discussed in Section 2, creating a dataset of about 1.8 million presentations. Approximately 1 million of these presentations are AC-equivalent to the presentations that remained unsolved by greedy search (c.f. Section 3). Only a small amount (roughly 15 percent) of the original Miller–Schupp presentations were part of this dataset.

The dataset is tokenized using six tokens: two stop tokens and one token each for the two generators and their inverses. The tokenized dataset had about 2.17×10^8 tokens. As our goal is to get insights into properties that distinguish GS-solved and GS-unsolved presentations, we performed an exploratory data analysis of the two subsets of data associated to these presentations. We plot the percentage of appearance of each token for these subsets in Figure 16. The ratio of frequency of $y^{\pm 1}$ to the frequency of $x^{\pm 1}$ is higher in the GS-unsolved dataset. This is likely because the GS-unsolved presentations have larger n, and larger n corresponds to a higher number of occurrence of $y^{\pm 1}$ in the Miller–Schupp presentation. Interestingly, this effect remains in the dataset even after applying thousands of AC-moves to the original presentations.

We paid special attention to ensure that our dataset contains presentations of a wide range of lengths so as not to bias our model towards learning trends specific to any fixed length. To this end, we devised an algorithm (Algorithm 6 in Appendix C) that creates an almost uniform distribution over the lengths of the presentations. (See Figure 17.) We set aside 10% of our entire data for validation.

7.3. **Results.** A randomly initialized model with the initialization scheme given in [Rad+19] has a cross entropy loss of $-\ln(1/n_{\text{vocab}}) \approx 1.7918$. With training, we could achieve a validation loss of 0.7337.[17] We used the untrained and the trained model to get the embeddings of all 1190 presentations of the Miller–Schupp series with $n \leq 7$ and $\text{length}(w) \leq 7$. We used t-SNE to project these embedding vectors to a plane [MH08]. The plots are shown in grid in Figure 18.

Each row of Figure 18 corresponds to a fixed value of n. The left (resp. right) column depicts t-SNE projections of embeddings obtained by an untrained (resp. trained) model. t-SNE dependence on a distance measure: it learns to map vectors that are closer together in the higher-dimensional space, with respect to this distance measure, close together in the plane [MH08]. We used cosine simiarity between embedding vectors as the distance measure for our plots. We note that the GS-solved and GS-unsolved presentations seem to cluster much more in the plots in the right column. This indicates that a trained Transformer model is able to distinguish

[17]We tuned the hyperparameters a little but it is quite likely that one can achieve a better performing model with more hyperparameter tuning. Similarly, more training data will necessarily help with the performance. We trained a Transformer model with hyperparameters given in Appendix A.

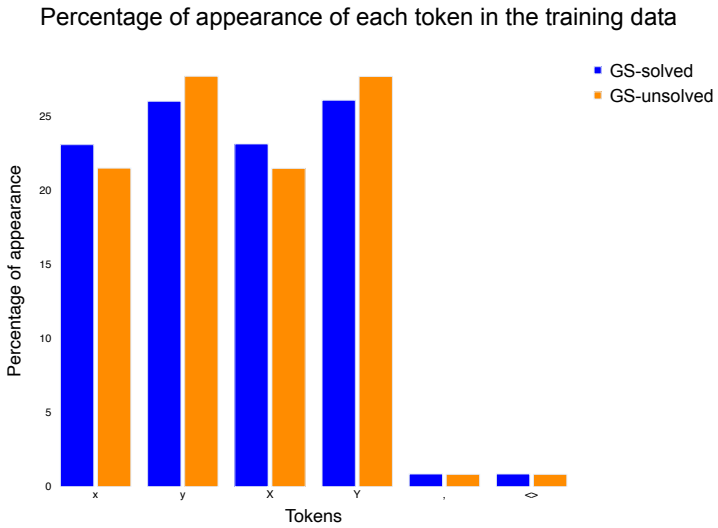


FIGURE 16. Percentage of appearance of each token in the two subsets of the training dataset that are equivalent to GS-solved and GS-unsolved presentations. To be clear, we computed the percentages separately for each subset of the training data, i.e. the heights of all blue (and orange) bars adds separately to 100.

between GS-solved and GS-unsolved presentations to a good extent, albeit not perfectly.[18]

Note that the training dataset contained no information about the ease of solvability of a presentation. It also did not contain many presentations of the Miller–Schupp series itself. Instead, it contained presentations that are AC-equivalent to the Miller–Schupp series presentations. Our observation that a Transformer model trained on this dataset can distinguish between the GS-solved and GS-unsolved presentations indicates that:

a) There likely exists an invariant at the level of the "language" of the balanced presentations that distinguishes GS-solved vs GS-unsolved presentations.
b) This invariant survives application of thousands of AC-moves we used to generate the training examples in our dataset.

APPENDIX A. HYPERPARAMETERS

Here we discuss the hyperparameters used to train the Proximal Policy Optimization (PPO) and Transformer models of Section 4 and Section 7 respectively. The hyperparameters of PPO are given in Table 1. These hyperparameters were

[18]Note also that t-SNE admits a hyperparameter known as "perplexity", and the projections learned by t-SNE depend on the hyperparameter [WVJ16]. Thus, in general, t-SNE plots must be interpreted with care. The plots shown in Figure 18 were all made with the perplexity value of 30. We checked however that the clusters of GS-solved and GS-unsolved presentations continue to exists at a broad range of perplexity values.

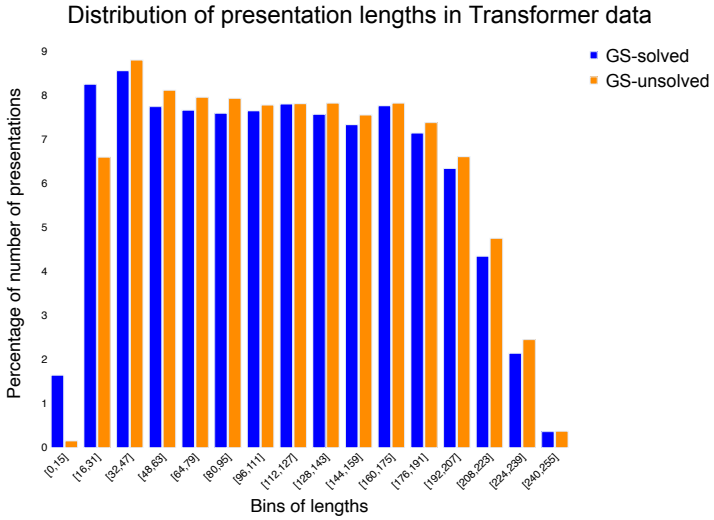


FIGURE 17. Percentage of presentations in various ranges of lengths. Percentages were computed independently for the two subsets of the dataset, corresponding to presentations that are AC-equivalent to GS-solved and GS-unsolved presentations. We used Algorithm 6 from Appendix C to ensure the almost-uniform distribution depicted here.

defined in the main text in Subsection 4.2. Each of the two networks —the actor and the critic— was a 2-layer feed forward neural network with 512 neurons and *tanh* non-linearlities. We used Adam optimizer for training.

The performance of PPO is known to be highly sensitive to various implementation details in addition to the choice of hyperparameters [Hua+22b; Eng+20]. We used the single-file implementation of PPO in CleanRL [Hua+22a], which has been well-benchmarked against the results of the original paper [Sch+17]. Following [Eng+20], we used advantage normalization and clipped value loss. If the KL divergence between the old and the updated policy exceeded the target KL divergence in a mini-batch, we skipped the remaining mini-batches in the optimization phase to avoid a large jump in policy. We did not ablate all of the choices of hyperparameters to investigate the importance of each choice.

The Transformer model studied in Section 7 is an 8-layer transformer model with the embedding space dimension of 512 and 4 attention heads. The context window of the Transformer had length 1024. We used a batch size of 12 and constant learning rate of 6×10^{-5}. We trained for a total of 25000 iterations. We used AdamW optimizer for training with hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We did not use any dropout during training.

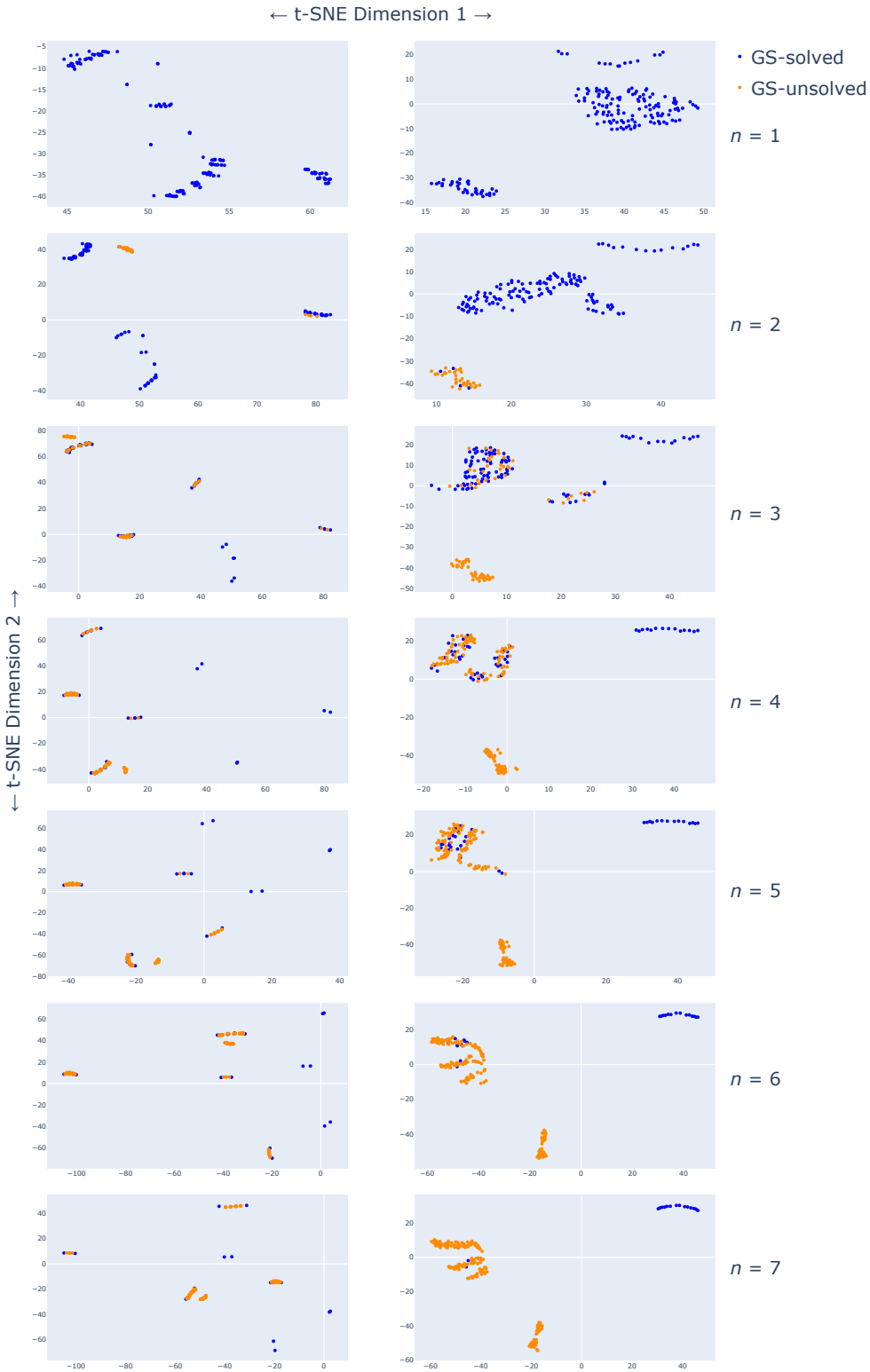


FIGURE 18. t-SNE plots depicting embeddings of the Miller–Schupp presentations $MS(n, w)$. The left (right) column shows embeddings learned by an untrained (trained) transformer model. Each row corresponds to a value of n. Trained models cluster together GS-solved and GS-unsolved presentations indicating the possibility of a difference in the linguistic structure of the two sets of presentations.

Hyperparameter	Value
Horizon (T)	200
Rollout Length (T')	200
Number of parallel actors	28
Total Number of Rollouts	$\sim 2 \times 10^5$
Maximum Learning Rate	1.0×10^{-4}
Minimum Learning Rate	0
Learning Rate Schedule	Linear Decay
Number of epochs	1
Number of mini-batches	4
Optimization mini-batch size	1400
Discount (γ)	0.999
GAE parameter (λ)	0.95
Clipping parameter ϵ	0.2
Value Loss coefficient, c_1	0.5
Entropy Loss coefficient, c_2	0.01
Adam epsilon parameter	10^{-5}
Target KL divergence	0.01

TABLE 1. Table of Hyperparameters

APPENDIX B. NEIGHBORHOOD CONSTRUCTIONS

B.1. **Neighborhoods of the identity.** For any $\ell \in \{3, \ldots, 16\}$, we constructed a neighborhood of the identity using an algorithm based on BFS search (Algorithm 4). This neighborhood contains all presentations that can be connected to the identity via a path of AC-moves, where each presentation in the path has a length less than or equal to ℓ, that is, the full based subgraph containing vertices with connectivity value less than or equal to ℓ. We consider the relators of a presentations as a set (meaning that the order of relators is not important; implemented as a tuple of relators in lexicographic order)

B.2. **Neighborhoods for MS series.** We define the n-neighborhood of a balanced presentation π as the set of all balanced presentations that can be obtained by applying at most n AC-moves to π. We used Algorithm 5, a variation of BFS, to generate 5-neighborhoods of presentations in the Miller–Schupp series. As before, we disregard the order of the relators.

APPENDIX C. LANGUAGE MODELING DATASET GENERATION

This appendix describes the method, Algorithm 6, used to generate the training and evaluation datasets for the Transformer model, as referenced in Section 7. Our aim was to create datasets featuring presentations of varying lengths. We began with a presentation P_0 from the Miller–Schupp series, where $n \leq 7$ and $\text{length}(w) \leq 7$, setting a maximum relator length $l_{\max} = 128$. Presentations were generated in $n = 128$ phases, each phase allowing a maximum relator length $l_i \sim \mathcal{U}(l + i \cdot l_{\text{inc}}, l + (i+1) \cdot l_{\text{inc}})$. Here, l represents the longest relator length in P_0 and $l_{\text{inc}} = (l_{\max} - l)/n$ is the incremental increase per phase. In each phase, we

Algorithm 4 Breadth-First Search Algorithm Bounded by Size

Input: A balanced presentation π, maximal size of presentation n
Output: Set of enumerated presentations connected to the starting presentation that are achievable without exceeding the size limit, and set of edges with filtrations
Initialize a queue Q, set of visited nodes *visited*, and numerical map *name* that will enumerate presentations
Mark π as visited, put it into queue Q, and assign it the number 0
while Q is not empty **do**
    $u \leftarrow$ top of Q ▷ Remove the front node of Q
    for every AC-move m **do**
        $child \leftarrow m(u)$
        if *child*'s size $\leq n$ and *child* is not visited **then**
            Put *child* in Q and mark it as visited
            Assign *child* the next available number
        end if
        if *child*'s size $\leq n$ and u's number is smaller than *child*'s number **then**
            Return edge $(u, child)$ with proper filtration
        end if
    end for
end while

Algorithm 5 Breadth-First Search Algorithm Bounded by Number of Steps

Input: A balanced presentation π, positive integer n
Output: n-neighborhood of π
Initialize a queue Q, set of visited nodes *visited*, and numerical map *dist* that represents the minimal number of AC-moves needed to transform π into a given presentation
Mark π as visited, put it into queue Q, and set its distance to 0
while Q is not empty **do**
    $u \leftarrow$ top of Q ▷ Remove the front node of Q
    for every AC-move m **do**
        $child \leftarrow m(u)$
        if $dist[u] < n$ and *child* is not in *visited* **then**
            Put *child* in Q and mark it as visited
            Set $dist[child] = dist[u] + 1$
        end if
    end for
end whilereturn set *visited*

selected a presentation P from the previous phase and applied $N = 1000$ AC′ moves. Any AC′ move that exceeded the length l_i resulted in no change.

We repeated this for all 1190 presentations in the Miller–Schupp series, ultimately producing approximately 1.8 million balanced presentations. The length distribution of these presentations is detailed in Figure 17.

Algorithm 6 Transformer Dataset Generation

Input:
  P_0 – an initial presentation with l as the length of the longest relator
  n – number of phases
  m – number of presentations in each phase
  N – number of AC′ moves to apply in each phase
  $l_{\max}$ – upper bound on presentation lengths in the dataset
Output:
  Dataset (the final collection of presentations)
Dataset $\leftarrow \emptyset$ ▷ Initialize the dataset of all presentations
$l_{\text{inc}} \leftarrow (l_{\max} - l)/n$ ▷ Increment for the maximum relator length per phase
for $i = 0$ **to** $n - 1$ **do** ▷ Loop over each phase
  for $j = 1$ **to** m **do** ▷ Generate m presentations for each phase
    $l_i \sim \mathcal{U}(l + i \cdot l_{\text{inc}}, l + (i + 1) \cdot l_{\text{inc}})$ ▷ Sample maximum relator length
    $P \leftarrow (i = 1)\ ?\ P_0\ :\ \text{Dataset}[(i - 1) \cdot m + j - 1]$
    for $k = 1$ **to** N **do** ▷ Apply N AC′ moves with relator length l_i
      $A \sim$ AC′ Moves
      $P \leftarrow A \cdot P$
    end for
    Dataset $\leftarrow$ Dataset $\cup\{P\}$ ▷ Add the presentation P to the Dataset
  end for
end for

Funding. The work of A.S. is supported by the US Department of Energy grant DE-SC0010008 to Rutgers University. The authors acknowledge the contributions of Office of Advanced Research Computing (OARC) at Rutgers University for providing access to the Amarel cluster and other computing resources. A.M.'s work is supported by NSERC grants RES000678 and R7444A03. A.M. also gratefully acknowledges the excellent working conditions provided by the Max Planck Institute for Mathematics in Bonn. The work of P.K. and B.L. is supported by the SONATA grant no. 2022/47/D/ST2/02058 funded by the Polish National Science Centre. This research was carried out with the support of the Interdisciplinary Centre for Mathematical and Computational Modelling at the University of Warsaw (ICM UW). The work of S.G. is supported in part by a Simons Collaboration Grant on New Structures in Low-Dimensional Topology, by the NSF grant DMS-2245099, and by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, under Award No. DE-SC0011632.

References

[AC65] James J. Andrews and Morton L. Curtis. "Free groups and handlebodies". *Proceedings of the American Mathematical Society* 16.2 (1965) (cit. on p. 5).

[Ach18] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. Online resource. 2018 (cit. on p. 14).

[AK85] Selman Akbulut and Robion Kirby. "A potential smooth counterexample in dimension 4 to the Poincare conjecture, the Schoenflies conjecture, and the Andrews–Curtis conjecture". *Topology* 24.4 (1985) (cit. on pp. 4, 6).

[Bag21] Neda Bagherifard. *Three-manifolds with boundary and the Andrews-Curtis transformations*. 2021 (cit. on p. 11).

[Ben+03] Yoshua Bengio et al. "A Neural Probabilistic Language Model". *J. Mach. Learn. Res.* 3 (Mar. 2003) (cit. on p. 29).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016 (cit. on p. 29).

[BM06] R. Sean Bowman and Stephen B. McCaul. "Fast searching for Andrews–Curtis trivializations". *Experimental Mathematics* 15.2 (2006) (cit. on p. 6).

[BM14] Jean-Daniel Boissonnat and Clément Maria. "The simplex tree: An efficient data structure for general simplicial complexes". *Algorithmica* 70 (2014) (cit. on p. 24).

[BM93] Robert G. Burns and Olga Macedonska. "Balanced Presentations of the Trivial Group". *Bulletin of the London Mathematical Society* 25 (1993) (cit. on p. 18).

[Bri15] Martin R. Bridson. "The complexity of balanced presentations and the Andrews–Curtis conjecture". *arXiv preprint arXiv:1504.04187* (2015) (cit. on p. 6).

[Bur+99] R. Burns et al. "Recalcitrance in groups". *Bulletin of the Australian Mathematical Society* 60.2 (1999) (cit. on p. 18).

[Cob+20] Karl Cobbe et al. "Leveraging procedural generation to benchmark reinforcement learning". *Proceedings of the 37th International Conference on Machine Learning*. ICML'20. JMLR.org, 2020 (cit. on p. 3).

[CV22] Gunnar Carlsson and Mikael Vejdemo-Johansson. *Topological data analysis with applications*. Cambridge University Press, Cambridge, 2022 (cit. on p. 24).

[Dou23] Michael R. Douglas. *Large Language Models*. 2023 (cit. on p. 28).

[DU24] Lennart Dabelow and Masahito Ueda. *Symbolic Equation Solving via Reinforcement Learning*. 2024 (cit. on p. 3).

[Elh+21] Nelson Elhage et al. "A Mathematical Framework for Transformer Circuits". *Transformer Circuits Thread* (2021). https://transformer-circuits.pub/2021/framework/index.html (cit. on p. 28).

[Eng+20] Logan Engstrom et al. *Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO*. 2020 (cit. on p. 32).

[HR] George Havas and Colin Ramsay. *Breadth-first search and Andrews–Curtis conjecture* (cit. on p. 6).

[Hua+22a] Shengyi Huang et al. "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms". *Journal of Machine Learning Research* 23.274 (2022) (cit. on p. 32).

[Hua+22b] Shengyi Huang et al. "The 37 Implementation Details of Proximal Policy Optimization". *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. 2022 (cit. on p. 32).

[IKS17] Hakan Inan, Khashayar Khosravi, and Richard Socher. *Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling*. 2017 (cit. on p. 29).

[KS16] Krzysztof Krawiec and Jerry Swan. *Distance Metric Ensemble Learning and the Andrews–Curtis Conjecture*. 2016 (cit. on p. 6).

[Lis17] Boris Lishak. "Balanced finite presentations of the trivial group". *Journal of Topology and Analysis* 9.02 (2017) (cit. on p. 6).

[Mar+14] Clément Maria et al. "The gudhi library: Simplicial complexes and persistent homology". *Mathematical Software–ICMS 2014: 4th International Congress, Seoul, South Korea, August 5-9, 2014. Proceedings 4*. Springer. 2014 (cit. on p. 24).

[MH08] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". *Journal of Machine Learning Research* 9.86 (2008) (cit. on p. 30).

[Mia03] Alexei D. Miasnikov. "Genetic algorithms and the Andrews–Curtis conjecture". *arXiv preprint math/0304306* (2003) (cit. on p. 6).

[MMS02] Alexei D. Myasnikov, Alexei G. Myasnikov, and Vladimir Shpilrain. "On the Andrews–Curtis equivalence". *Contemporary Mathematics* 296 (2002) (cit. on pp. 7, 11).

[MS99] Charles Miller and Paul Schupp. *Some presentations of the trivial group*. 10.1090/conm/250/03848. 1999 (cit. on p. 6).

[MSZ16] Jeffrey Meier, Trent Schirmer, and Alexander Zupan. "Classification of trisections and the generalized property R conjecture". *Proc. Amer. Math. Soc.* 144.11 (2016) (cit. on p. 11).

[MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. "Linguistic Regularities in Continuous Space Word Representations". *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Ed. by Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff. Atlanta, Georgia: Association for Computational Linguistics, June 2013 (cit. on p. 29).

[OAC23] OpenAI, Anthropic, and Cohere. *Needle In A Haystack — Pressure Testing LLMs*. Online resource. 2023 (cit. on p. 3).

[PG23] Gabriel Poesia and Noah D. Goodman. "Peano: Learning Formal Mathematical Reasoning". *Phil. Trans. R. Soc.* A.381.2251 (2023) (cit. on p. 3).

[PU19] Dmitry Panteleev and Alexander Ushakov. "Conjugacy search problem and the Andrews–Curtis conjecture". *Groups Complexity Cryptology* 11.1 (2019) (cit. on p. 6).

[PW17] Ofir Press and Lior Wolf. *Using the Output Embedding to Improve Language Models*. 2017 (cit. on p. 29).

[Rad+19] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". 2019 (cit. on p. 30).

[Sch+17] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017 (cit. on pp. 4, 14, 32).

[Sch+18] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018 (cit. on p. 14).

[Tau+21] Guillaume Tauzin et al. "giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration". *Journal of Machine Learning Research* 22.39 (2021) (cit. on p. 24).

[Tri+24] Trieu H Trinh et al. "Solving olympiad geometry without human demonstrations". *Nature* 625.7995 (2024) (cit. on p. 3).

[Vas+23] Ashish Vaswani et al. *Attention Is All You Need*. 2023 (cit. on p. 28).

[Wad94] Masaaki Wada. "Twisted Alexander polynomial for finitely presentable groups". *Topology* 33.2 (1994) (cit. on p. 12).

[WVJ16] Martin Wattenberg, Fernanda Viegas, and Ian Johnson. "How to Use t-SNE Effectively". *Distill* (2016) (cit. on p. 31).

A.S. — NHETC, Department of Physics and Astronomy, Rutgers University, Piscataway, New Jersey 08854, USA

A.M. — Department of Mathematics, Western University, ON, Canada

B.L. — Institute of Mathematics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland

A.G. — Polygon Zero

P.K. — Institute of Mathematics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland

S.G. — Richard N. Merkin Center for Pure and Applied Mathematics, California Institute of Technology, Pasadena, CA 91125, USA