Solving Abstract Reasoning Tasks with Grammatical Evolution

Raphael Fischer, Matthias Jakobs, Sascha Mücke, and Katharina Morik

TU Dortmund, AI Group, Dortmund, Germany
http://www-ai.cs.tu-dortmund.de

Abstract. The Abstraction and Reasoning Corpus (ARC) comprising image-based logical reasoning tasks is intended to serve as a benchmark for measuring intelligence. Solving these tasks is very difficult for off-the-shelf ML methods due to their diversity and low amount of training data. We here present our approach, which solves tasks via grammatical evolution on a domain-specific language for image transformations. With this approach, we successfully participated in an online challenge, scoring among the top 4% out of 900 participants.

Keywords: Machine Learning · Reasoning · Grammatical Evolution

1 Introduction

Despite AI research's fast advancements, the question of how to rigorously define and measure intelligence is still open [5,3]. It is taken up by the recently published *Abstraction and Reasoning Corpus* (ARC) [1] and its corresponding *Kaggle* challenge[1]. It features hundreds of image-based logic tasks (some examples given in Figure 1), which are expected to be solved by reasoning AIs without any human aid. Finding solutions requires learning the inherent logic of a task from very few examples, which is easy for humans but proves to be very hard for machines. Learning logic from few examples has already been explored (e.g. for text data [2]), however ARC's image logic space is much larger.

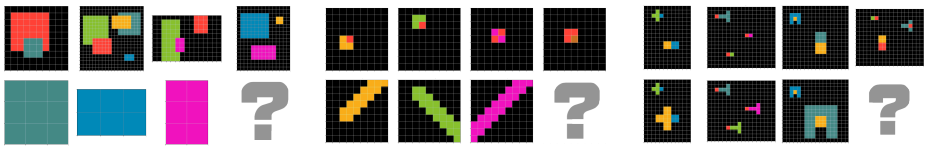
We here present our approach to solve abstract reasoning tasks based on a *domain-specific language* (DSL), whose expressions are generated via *grammatical evolution* (GE). With our method, we were able to reach the 28th place out of over 900 participating teams in the ARC challenge[2]).

2 Problem Statement

The ARC challenge requires participants to develop a model that is able to solve tasks D_i from the set $\mathcal{D} = \{D_1, \ldots, D_M\}$. Each D_i comprises an image-based

[1] https://www.kaggle.com/c/abstraction-and-reasoning-challenge/

[2] *ls8-arc* team on the official ARC challenge leaderboard.



(a) Training task 48 (b) Training task 100 (c) Training task 397

Fig. 1: Exemplary ARC tasks: (a) Crop to the smallest unicolor rectangle; (b) Draw lines over the image in directions indicated by pixels colored in red; (c) Match pattern, upscale and/or recolor if necessary.

reasoning task with m_i training pairs $\{(x^k, y^k)\}$ and n_i test pairs $\{(\hat{x}^\ell, \hat{y}^\ell)\}$, where $m_i > 0$ is typically around 3 and $n_i > 0$ is mostly 1. The images feature up to 10 discrete colors $C = \{c_1, \ldots, c_{10}\}$, and image sizes range between 1 and 30 pixels per dimension. The implied function f that maps inputs to the expected output images is vastly different for every task, often based on abstract features (e.g. connected shapes) or pattern continuation (see Figure 1). For each task D_i, the method should derive $\tilde{f}_i$ from $\{(x^k, y^k)\} \in D_i$ which correctly maps all input images to the expected output: $\forall(\hat{x}^\ell, \hat{y}^\ell) \in D_i : \tilde{f}_i(\hat{x}^\ell) = \hat{y}^\ell$.

The challenge participants have access to $M = 400$ training tasks, which show some logic concepts and come with labels $\hat{y}^\ell$. The final scoring however is based on an undisclosed test set, whose task are only seen by the model.

3 Reasoning Approach

For our approach, we assume that f can be broken down into a sequence of basic image transformations. We developed a custom *domain-specific language* (DSL) specifying such sequences, whose space of expressions is explored using an *evolutionary algorithm* (EA). We first explain our language in more detail and then show how we use an EA to create ARC task solvers from our DSL.

Domain-Specific Language In a first step, we manually implemented solvers for approximately 20 random training tasks and identified reoccurring image operations, which became the basis of our DSL.

Let $\mathcal{X} = \bigcup_{u,v \geq 1} C^{u \times v}$ denote the set of rectangular images with colors in C, and $\mathcal{X}^*$ ordered lists of images, which we call *layers*. Our DSL is a context-free grammar in *Backus Naur Form* (BNF) [6]; the non-terminal symbols represent function sets whose members (i) modify images ($T = \mathcal{X}^\mathcal{X}$), (ii) decompose an image into layers ($S = (\mathcal{X}^*)^\mathcal{X}$), (iii) combine layers into a single image ($J = \mathcal{X}^{\mathcal{X}^*}$) or (iv) modify a layer object ($L = (\mathcal{X}^*)^{\mathcal{X}^*}$). The terminal productions of these symbols are either concrete functions of the corresponding type or more complex function compositions. Figure 2 depicts a visualization of our DSL function types.

Our atomic functions comprise basic operations such as translation, rotation and cropping, as well as layer-specific operations like extracting a layer, sorting

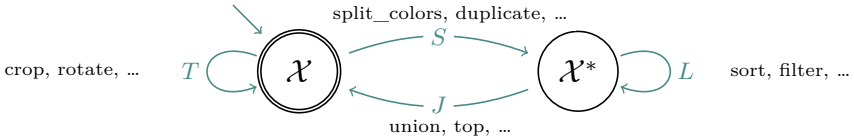


Fig. 2: Function types of custom reasoning DSL, displayed as state automaton, with some example functions for each type.

by different criteria, splitting images into layers, and merging them together. For additional flexibility, we also allow higher-order functions like *map*, which lifts the T-type to an L-type function by applying it to each layer. The grammar structure ensures that all possible expressions have an overall T-type logic, i.e. they transform a single input image into an output image. This allows to find solvers for some tasks, an example is given in Figure 3.

Grammatical Evolution We use Grammatical Evolution (GE) [7] to generate expressions in our DSL, and ultimately find solvers for a given task. We use the standard modulo-based mapping from codons to syntax trees of our DSL [6] to obtain image functions $\tilde{f}$. Uniform mutation and 1-point crossover are used to produce offspring [4]. To prevent running out of codons, we limit the maximum tree depth by preemptively excluding rules at every tree node.We choose the next generation's parents via tournament selection on the combined parent and offspring population. For this we assess the loss value of each function, given by the distance of its outputs $\tilde{f}(x)$ to the ground-truth output images y, averaged over all m_i training pairs,

$$\mathcal{L}(f|D_i) = \frac{1}{m_i}\sum_{k=1}^{m_i} d_{\text{img}}(f(x^k), y^k). \tag{1}$$

Here we define d_{img} as (i) the proportion of correctly colored pixels (if images have equal size) or (ii) the Euclidean distance between the color histograms:

$$d_{\text{img}}(x,y) = \begin{cases} \sum_i \mathbf{1}_{\{x_i \neq y_i\}}/(u_x v_x) & \text{if } u_x = u_y \text{ and } v_x = v_y \\ 1 + \|\phi(x) - \phi(y)\|_2 & \text{else} \end{cases} \tag{2}$$

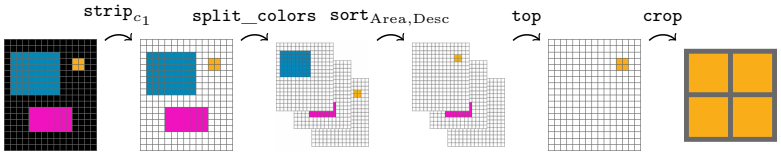


Fig. 3: Solver for task displayed in Figure 1a. It first strips away black (c_1) pixels, then splits the image into layers according to the remaining colors, and finally crops to the layer with smallest surface area.

where u, v is the image width and height, $\mathbf{1}_{\{P\}} = 1$ if $P \equiv \top$ else 0, and $\phi(x) = \sum_{x_i \in x} (\mathbf{1}_{\{x_i = c\}})_{c \in C}^{\top}$ as the (unnormalized) color histogram of x. $\tilde{f}$ is an optimal solution if (and only if) all predictions are equal to the expected output, i.e. $\mathcal{L}(f|D_i) = 0$, thus we can also use Equation 1 as an early-stopping criterion.

4 Experimental Results

Following the evaluation procedure of the challenge, our accuracy is the proportion of correctly solved tasks: $\text{ACC} = M^{-1} \sum_{i=1}^{M} \prod_{\ell=1}^{n_i} \mathbf{1}_{\{f_i(\hat{x}^\ell) = \hat{y}^\ell\}}$. Here, f_i is the solution produced by our method after training on task D_i. A grid search was performed to obtain good hyperparameters for population size, mutation rate and mutation strength. As GE is randomized, we ran the experiments with 40 different seeds and discuss the averaged results with standard deviation.

We first evaluate our method on the 400 training tasks of ARC, from which we solved 7.68($\pm$0.61)%. The challenge leaderboard evaluation however is based on a secret data set of 100 tasks with slightly higher logic complexity. Here, we were able to correctly solve ACC = 3% of the tasks. Despite this seemingly low value, we scored among the top 30 of over 900 participants, which illustrates the challenge's non-triviality. The small number of tasks makes it hard to assess the usefulness of atomic functions in the DSL. Moreover, there is no information about the overlap of required logic operations between training and test tasks.

Our results also made us question the effectiveness of GE considering the tremendous search space complexity. Small mutations to the current best individual, such as swapping out a single atomic function, can significantly change its behavior. As a result, the algorithm operates in a highly non-convex search space. We therefore compared our EA approach to simply generating random individuals from our DSL. This random search baseline solves an average of 6.17($\pm$0.13)% of the training tasks, indicating that the EA is able to traverse the search space at least somewhat more efficiently. This is even more significant on the test tasks, where random-search is not able to solve even a single task.

5 Conclusion

We showed that our DSL+GE approach is viable for solving reasoning tasks. By designing a language for image-based logic and learning corresponding expressions from the training instances, we were able to score well in the ARC challenge. Ensuring a high expressiveness of the DSL, while at the same time limiting the complexity to allow tractable function evolution, appears to be the key problem. Finding the optimal set of functional atoms for the given domain may thus be subject to further research.

The ARC challenge's difficulty illustrates the long way still to go until ML methods reach abstract reasoning capabilities comparable to the human mind's.

Acknowledgment This research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01|18038A)

References

1. Chollet, F.: The measure of intelligence. arXiv preprint arXiv:1911.01547 (2019)
2. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices **46**(1), 317–330 (2011)
3. Hernández-Orallo, J., Martínez-Plumed, F., Schmid, U., Siebers, M., Dowe, D.L.: Computer models solving intelligence test problems: Progress and implications. Artificial Intelligence **230**, 74–107 (2016)
4. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and Computing **4**, 87—112 (1994)
5. Legg, S., Hutter, M.: A collection of definitions of intelligence. Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms **157** (07 2007)
6. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation **5**(4), 349–358 (2001)
7. Ryan, C., Collins, J.J., Neill, M.O.: Grammatical evolution: Evolving programs for an arbitrary language. In: European Conference on Genetic Programming. pp. 83–96. Springer (1998)