Efficient learning through programmatic representations

How can we better find the matches between input and output objects?

Alexandra-Laura Zaghăr[1]

Supervisor(s): Sebastijan Dumančić[1]**, Dekel Zak**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Alexandra-Laura Zaghăr
Final project course: CSE3000 Research Project
Thesis committee: Sebastijan Dumančić, Dekel Zak, Arie van Deursen

Abstract

The Abstraction and Reasoning Corpus (ARC) serves as a challenging benchmark designed to measure human-like artificial intelligence by only providing a few training examples for each task. Program synthesis offers a new approach to solving this benchmark by generating rule-based transformation programs. A recent approach, BEN, tackles these tasks by making use of program synthesis in a Divide, Align and Conquer strategy. The Align component identifies correspondences between the input and output objects by using a Structure Mapping Engine (SME) rooted in analogical reasoning. Despite its success, each individual part of the algorithm can be further improved, in particular the features of objects used in Align.

We present an enhanced object-matching methodology of the Align component to improve the quality of the correspondences found. First, the BEN algorithm is re-implemented in Julia, making use of an Answer Set Programming (ASP) solver using Clingo and Prolog for the SME to offer better efficiency. Second, we augment the structural representation of the objects by introducing features that capture the spatial relations between them, specifically through forms of ranked coordinates. Furthermore, we refine how multi-coloured objects are propositionally encoded. Finally, we introduce a weighting heuristic for the features: the significance of individual visual attributes is minimized when the input and output grids contain the same number of objects, and simple coordinates are eliminated when the input and output grids have different sizes.

The proposed changes were evaluated by isolating the Align component across subsets of the ARC-AGI-1 benchmark. In a sample of 25 manually selected tasks, the number of perfectly matched tasks improved significantly from 11 to 23. In a randomly selected sample of 25 tasks, the modifications give better or identical matches in 22 tasks, with only 3 showing degradations. When running the entire benchmark with the full BEN algorithm, one extra task is solved and two no longer are. Average times are similar and the number of searched transformations decreases. These findings suggest that including spatial relations and contextual weighting of the features improves the accuracy of finding correct correspondences for the ARC benchmark.

1 Introduction

The Abstraction and Reasoning Corpus (ARC) is a benchmark that was made to challenge AI models. Instead of having a significant amount of training data, this benchmark offers only 2-4 examples per task, with the scope to better gauge the level of ”intelligence” AI is capable of without giving it an unlimited amount of prior knowledge [1]. An example of such a task from the benchmark can be seen in Figure 1. It provides inputs and outputs in the form of grids, each task having only a couple pairs of input-output training examples. It is an especially interesting benchmark since some of the better agents previously managed to only solve 20% of the tasks with a brute force approach and domain specific languages [2] [3].

The ARC benchmark can be interpreted from several perspectives: as a benchmark for general artificial intelligence, as a program synthesis task, and as a psychometric test of intelligence [4]. When viewed as a program synthesis problem, where the objective is to generate a program satisfying a given specification from a given grammar [5], a possible approach is to use the provided examples to synthesize a program that determines when and how specific transformations should be applied.

Using this, a new, better way of tackling the problem was made using a Divide, Align, and Conquer approach [6], specifically in regards to the first version released by ARC in 2019. The paper outlines the 3 different steps of the algorithm, BEN. Firstly, in the Divide stage, the input and output are split into separate objects based on several segmentation rules. Then, the Align part matches objects between them. The purpose of this step is to yield pairs of input/output objects that can be transformed from the former into the latter using a program. This step is done based on the structural alignment of the grids, using various features of the objects. Finally, the Conquer stage is made up of two distinct parts. The first one is finding the transformations for the pairs of objects given from the Align step. Then, based on these transformations, a concept is learned of when to apply them to the inputs based on the object having or not having certain features.

Although this approach significantly improves the performance on the ARC tests, each individual part can be further improved. Since there were no sufficiently similar baseline solutions for comparison, some of the design decisions can now be refined by using BEN as the reference baseline. This paper will focus on the second step of the algorithm - the Align component, which matches input and output objects. This is done by using a structure mapping engine, which was inspired from analogical reasoning [7]. While the solution already makes use of 15 different attributes of the objects, such as colour, shape, and size, which attributes are used can be further improved. Because of the variety in the ARC tests, these features can interact in unexpected ways when trying to find the correct matches. This is an important step in the algorithm, since the first synthesized transformations are derived from the highest-ranked matches. When the correct correspondences appear near the top of the list, the algorithm can generate more accurate transformations earlier in the process, which accelerates convergence toward the final solution. As a result, the overall efficiency can be improved, because fewer incorrect or low-quality matches need to be explored.

Therefore, the research question posed is: How can we better find the matching objects from the inputs and outputs? This paper expands on what heuristics could be used to decide what features are more important in different cases, as well as

what features that were not considered previously could improve the matches.

There are two main options explored. Firstly, we added features that express spatial relations between objects and changed how colour is encoded. Secondly, we added two heuristics. When the number of input and output objects is equal, features such as size, width, and height are less important. We claim that giving them a lower significance does not negatively impact the tasks where these attributes are still shared between matched objects, as taking into account the spatial relations is enough to still give correct correspondences. Moreover, when the grid sizes are different, the specific coordinates no longer match between the inputs and the outputs, thus we remove them.

The results indicate that the options explored improve how well the matches are found. When testing Alignment in isolation, in a sample of 25 manually selected tasks, the changes improved performance from 11 tasks with all matches correctly identified to 23 tasks. In a sample of 25 randomly selected tasks, the changes improved matches found in 5 tasks, with only 3 tasks showing a reduction in correspondences found. On the entire benchmark, one additional task was solved while two others were no longer solved. The average times needed both for searching transformations and for total solving time remain similar. The average number of searched transformations decreases from around 18170 to 17320.

2 Background

Our work focuses mainly on improving how well the matches between input and output objects are found in the context of using BEN to solve the first benchmark released by ARC, called ARC-AGI-1.

ARC benchmark

The existence and rise of AI created a very interesting problem: what is the actual definition of intelligence? How can we measure the capacity a machine has for intelligence in the way a human perceives it?

A significant number of current AI models share a common starting point: vast amounts of data. Amounts such that a human could never process even a small percentage of it during their lifetime. Thus, we can question if just testing the skills exhibited while ignoring the amount of prior knowledge and experience is really testing intelligence.

Chollet [1] posed this exact problem and came up with a benchmark that can test a different type of intelligence than current AI models showcased. The benchmark is made up of 400 training tasks and 600 evaluation tasks. In turn, each task has, on average, 3.3 demonstration examples and one test example. A visualization of one such task can be seen in Figure 1.

Each example given has an input grid and an output grid of size maximum 30 by 30. The entire idea of the benchmark is to try and assess the capability of learning from a small number of training inputs. In the given example, the rule that should be learned is to fill the gray boxes with the same pattern as the coloured one in the input.

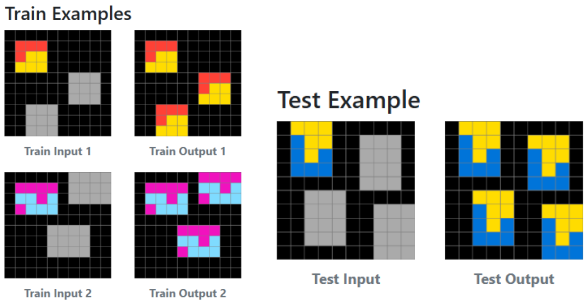


Figure 1: Visualization of ARC task d364b489. Adapted from [8].

Program synthesis and BEN

Considering the nature of the tasks, one way to approach solving this benchmark is with program synthesis - the task of generating a program satisfying a given specification from a given grammar [5].

One algorithm that leverages program synthesis is BEN [6], an algorithm structured using Divide, Align, and Conquer.

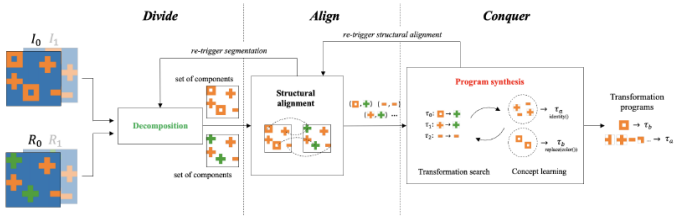


Figure 2: Structure of BEN, taken from [6]. It shows the three main steps in BEN: Divide, Align, and Conquer.

The first part, Divide, receives as input the different images in the task. Each input and each output image are grids of maximum 30 by 30 pixels. Each grid is composed of multiple objects and a background colour. This part of the algorithm segments the grids into their respective objects. Here the different rules for how objects are defined are selected. These are related to how many colours are allowed in the same one, or if diagonal pixels are considered in the same object. The background colour is also selected in this part. In the example in Figure 2, the segmentation would result in orange and green objects that have either a square- or plus-like shape.

Secondly, the Align part tries to match the input and output objects. This is done using the structural alignments of the grids, which are encoded based on certain features of the objects. This is described in more detail in the next subsection. In the given example, orange squares in the input will be matched with the green pluses, based on spatial features.

Lastly, the Conquer part is made up of two steps. The first is finding necessary possible transformation for the pairs given by Alignment. Then, a concept is learned over each transformation and the correspondences it covers.

The former part is achieved by enumerating possible transformations up to a fixed depth, with some of the transformations already determined by the specific pair of objects under consideration. In the correspondences given in Figure 2, the transformations found could be identity, as well as change shape to plus and colour to green.

Lastly, to learn the conditions for applying transformations, input objects are divided into three groups for each transformation: positive examples, which correctly generate an output object; negative examples, which incorrectly generate output parts; and neutral examples, which generate neither completely correct nor incorrect outputs. These groups are then used to learn which features determine when each transformation should be applied. In the example in Figure 2, the transformation rules found would be to change shape to plus and colour to green if the object is square and orange, and apply just identity otherwise.

Align - Structure Mapping Engine

The implementation of the Align part of the algorithm is strongly based on analogical reasoning, which is a central part of human cognition [9]. Within this field, one of the most widespread theories developed is the structure-mapping theory [10]. It argues that analogy is the transfer of the relational structure from a base to a target, and that relations between the objects are more important that just their attributes in isolation. Based on this theory, the Structure Mapping Engine (SME) [7] was built, which searches for matches that maximize structural alignment.

In BEN, after the Divide part, each object is described through 15 different features that capture its most relevant attributes. These are used to represent each object using a propositional encoding scheme, which includes predicates for each object. The arguments for each of them represent the features of the object.

Then, the objects are matched using SME, which consists of three steps: creating match hypotheses, expanding them into global mappings, and then ranking these. To explain how the engine works in more detail we will look at how it acts on a simple example, seen in Figure 3. The base is represented by water that flows from a beaker to a vial via a pipe, action caused by a greater pressure in the beaker than in the vial. As additional information, the diameter of the beaker is greater than the vial's, water is a liquid and it has a flat-top. In the target, heat flows from coffee to an ice-cube via a bar. We also know that the temperature of the coffee is greater than the ice-cube's, and that coffee is a liquid and has a flat-top. An additional, less technical visualization of this example can be seen in Figure 15 Appendix A.1.

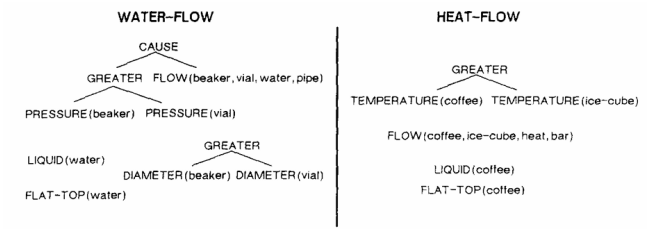


Figure 3: Vizualisation of the encoding of the water-flow and heat-flow analogy example, taken from [7].

The first step in SME is to construct *local matches*, for example between pressure and temperature, that later turn into *match hypotheses* of the items in the base and the target based on certain rules. One such rule could be that if the created match hypothesis is between two expressions with arguments, matches will be created between all corresponding arguments of the two. A complete set of rules can be seen in [7]. Running a match constructor based on these rules results in a set of correspondences.

In the example from Figure 3, greater temperature in the target will appear in two match hypotheses, one with greater pressure and one with greater diameter. Given a rule that a greater or a flow predicate can only be matched with another predicate of the same name, these will never be matched with pressure, for example. However, temperature will be matched with both pressure and diameter. Matching the flow in base and target will then lead to match hypotheses between beaker and coffee, vial and ice-cube, water and heat, as well as pipe and bar.

The second step of SME is creating *global mappings* (GMAPs). A GMAP is a structurally consistent and maximal set of match hypotheses. Structurally consistent means that no item in the base is matched to multiple items in the target or vice versa. In addition, if a match hypothesis is in a GMAP G, all match hypotheses that include its arguments are also in G. Moreover, each GMAP is maximal: adding any additional match hypotheses would make it no longer be structurally consistent.

In the example given, there would be three GMAPs generated by the engine. The first would follow from greater pressure being matched to greater temperature and would include all four matches of items: beaker with coffee, vial and ice-cube, water and heat, pipe and bar. The second GMAP would include only beaker with coffee and vial with ice-cube, following from greater diameter being matched to greater temperature. The final GMAP would only have the match between water and coffee, from the liquid and flat-top predicates.

The final step in SME is to rank the generated GMAPs. Each individual match hypothesis is assigned a score. The intuition behind these scores is that each match hypothesis is treated as a form of "evidence" and the score represents a "degree of belief." A match hypothesis will have a higher score if there is more structure above it that is matched. A complete explanation on how they are assigned can be found in [7]. The overall score of a GMAP is simply the sum of all the scores of the match hypotheses in it.

From the three generated GMAPs in our example, the one with the highest score will be the one that covers all items in the flow predicate. Then, the second ranked one only contains beaker with coffee and vial with ice-cube, while the last one will be the one with just the match between water and coffee.

3 Methods

The first part of this project was re-implementing the existing algorithm in Julia, which was the only part done with the entire team. Then, we tested how well the improvements would do in multiple ways. Apart from running the entire benchmark, two samples were chosen to test Alignment in isolation.

The first sample is made of 25 tasks that were manually selected. These contained both tasks that we would expect the current implementation to do well on and to fail on, to ensure

the changes made would not negatively impact the performance. The improvements chosen were based on this sample to improve the number of matches that are correctly found.

Then, to test how the changes would impact the rest of the benchmark, we randomly sampled another 25 tasks from the benchmark. To ensure their relevance for testing just Alignment in isolation, some tasks were skipped. In particular, those with only one object in input and those that are conceptually hard to segment were not included. Some examples of tasks we would consider not relevant can be seen in Appendix A.2.

We encoded all 50 tasks manually into JSON files. These contained both the attributes of the objects, as well as what the expected matched would be. Both of them were chosen based on what would be the most plausible selection when solving the task as a human, not necessarily on what the segmentation part would give or could give without improvements.

To evaluate the impact of the changes on the full algorithm, all 400 tasks were used. The baseline was the re-implemented version of BEN, including the improvements to both components of the Conquer phase.

3.1 Re-implementation

The first step was to re-implement the BEN algorithm in Julia in order to make use of Herb.jl [11], a library that makes using program synthesis easier in the Conquer step of the algorithm.

To ensure the main considerations for the Align part are the features used and not the speed, the ASP solver Clingo [12] is used, along with an implementation of the SME in Prolog [13]. This significantly outperforms the initial Python implementation of the engine, thus allowing for more attributes to be used in encoding without drawbacks on performance.

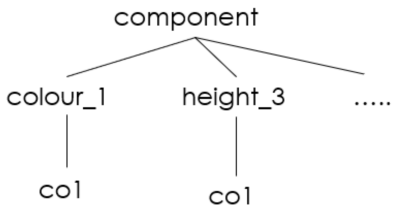


Figure 4: Incomplete new encoding of an object with colour 1 and height 3.

The first change that needed to be done was slightly alter the way features are used in encoding to make it work with the SME implementation. Considering the given implementation checks structural alignments based on predicates, each object is a predicate with one argument for each of the features used. Then, each feature is itself a predicate with one attribute - the object it applies to. Each combination of feature and value of it has its own predicate, so that the structural alignment is better when objects share the same value for a feature. In Figure 4 there is the incomplete encoding of an object that has colour 1 and height 3.

3.2 Improvement to features

There are two main approaches we took for improving the Alignment step. The first one was looking into what features could be added to improve how well the matches are found. The second was devising heuristics of when certain features should be less important or not included at all.

Features used

There are multiple aspects to consider about the way humans make analogies between objects, which shape what features are better to use in the Align step.

Firstly, there are two levels of analyzing a picture that contains objects: at a global level or at an object level. The global level entails focusing more on the overall picture, how an object is related to the others, while the object level has individual features of the objects as the center of attention. Which one is used affects how the connections between objects are made in the end. People tend to look at a global level more often than just at the details of the objects, especially if the way they are placed is salient [14]. In the initial implementation of the algorithm, the features used to represent a global rather than object level view of the grid are the rankings of colour, shape, and size. A rank encodes how common each attribute value is across all objects in the grid, ordered from most to least frequent. These offer insight into the regularity of the overall grid and we call global level features.

Another aspect to consider is that of spatial relations between objects. A study done showed that a model that used a structural representation with both spatial relations between objects and their visual features achieved close to human accuracy on a mapping task [15]. In the initial implementation of BEN, spatial information about object locations is represented solely through the row and column coordinates of the top-left corner of the bounding box. This is quite limited and does not give any information about relations between the objects due to how encoding is done. Therefore, we added a ranked version of both the x and the y coordinates. This rank gives the order from left to right of the objects, as well as from up to down. Moreover, to include tasks similar to that in Figure 5, we added four additional predicates to link the different ranked coordinates: matching the horizontal order of the input object to the vertical one of the output (and vice versa), as well as matching each axis to its reversed version. This allows for matching based on mirroring as well. This category of features are the spatial relations one.

Finally, we have the last category of attributes: visual, isolated features of the objects. This category has the following features from the initial implementation: colour, number of colours, size, width, height, shape, and whether the object is filled. A study found that colour similarity is stored in the early visual areas of the brain and arrives in the visual cortex faster than shape-related features [16], making it possibly more important than other isolated features. However, in the initial implementation, colours are encoded as just a string representation of the list of colours the object is made out of. This means that objects such as those in Figure 6 are seen as having completely different colours, even though a human would perceive them as very similar. Therefore, we added two more predicates to encode colour: one that is used if the

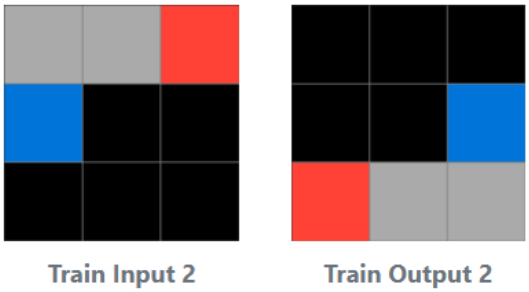


Figure 5: Visualization of ARC task 6150a2bd. Additional ranked versions of the x and y coordinates are needed for it. Adapted from [8].

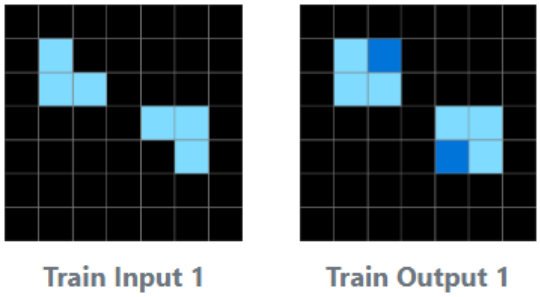


Figure 6: Visualization of ARC task 3aa6fb7a. Initial colour encoding would see these as completely different. Adapted from [8].

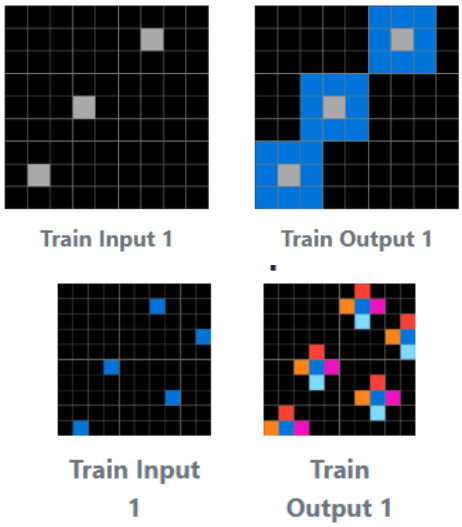


Figure 7: Visualisation of tasks 4258a5f9 and d364b489. For these, the spatial relation between objects matters, while attributes such as width, height, or top left corner are not relevant anymore. Adapted from [8].

objects share at least one colour, and one that signifies the objects share at least half of their colours.

All the features used for encoding the objects can be seen in Table 1.

Heuristics

Another approach to improving the algorithm is searching for heuristics of when some attributes are more important than others. In structure-mapping theory, one of the core mapping principles is that relations between objects are more important than their individual attributes [10]. In the context of ARC, the features in the third category are still important, but not in all tasks. In Figure 7, there are two examples of such tasks. We noticed in general that for many tasks where the number of input and output objects is the same, width, height, size, and top left corner no longer make sense, and the ranked coordinates are more important. Therefore, as a heuristic, we give different weights to the attributes when the number of input and output objects is the same: width, height, size, and top left corner have a weight of 1, ranked coordinates have a weight of 3, and everything else a weight of 2. Moreover, when the sizes of the grids are different from input to output, the top left coordinates are just removed from the encoding, as they almost never make sense.

Category	Attributes
Global level features	ranked shape, colour, size
Spatial relations	x, y, ranked coordinates, ranked-to-reverse-ranked coordinates, ranked-to-opposite-ranked coordinates
Visual, isolated features	colour, number of colours, size, width, height, shape, filled, sharing some colours, sharing more than half of the colours

Table 1: All features used for object encoding

4 Experimental setup

There are 3 different types of tests that were conducted to evaluate the performance of the changes made. The first is using the entire benchmark and running the full BEN algorithm on the 400 tasks. The other two experiments test Alignment in isolation. The second one makes use of a sample of 25 manually chosen tasks, while the third uses a sample of the same size of randomly chosen tasks.

All experiments were conducted on a machine that has a 13th Generation Intel Core i7-13700H processor (14 cores), 16 GB of RAM, and running Microsoft Windows 11.

For the first experiment, the ARC-AGI-1 public training set is used. For this, we measured how many tasks are successfully solved within the given timeout of 120 seconds both before and after the changes done to matches. Moreover, we measured the average time spent in finding the transformations and the average time needed to solve a task. Only those that do not time out are taken into account. Furthermore, we look at how many transformations were searched on average.

The ablation study done by Witt et al. [6] shows the somewhat limited influence the Align part has on the number of tasks solved. Therefore, smaller changes such as the ones we implemented might not be easily visible when testing the entire algorithm. As such, two more types of experiments were done to measure the impact on how well matches are found. Both of these test the Align part in isolation, using as input the objects in the grids and as output what the best matches should be.

There are no available datasets for the objects in the ARC benchmark. Therefore, for this testing, any example chosen

Metric	Before	After
Average solving time	44.13	46.29
Average time spent searching transformations	1.01	0.95
Average number of transformations searched	18173.9	17319.5

Table 2: Comparison of results on the 400 tasks in the ARC-AGI-1 benchmark before and after the changes.

had to be manually encoded, both for the input and expected output, which were chosen by us. As such, only a subset of the 400 given tasks were used.

Firstly, 25 examples were manually chosen for their relevance to the changes made. These include the cases specifically targeted for improvement - tests where the input and output images have the same number of objects, or different sized grids. Some of the changes implemented were chosen based on this sample to improve the performance. The manual selection was done in such a way that tests that both pass and fail initially are present, to ensure that the changes do not negatively impact working instances. For these, we look at number of passing and failing tests, where passing means finding all the correspondences as expected.

Secondly, the effect to non-targeted examples is tested. For this purpose, a sample of 25 cases was randomly chosen from the rest of the benchmark. To ensure relevance, some tasks were skipped: those that have only one or no input object, as well as those that would be hard to conceptually segment into objects. These tests are put into three categories: tests where the changed algorithm performs the same, better, or worse. We measure the performance based on the number of correct correspondences found.

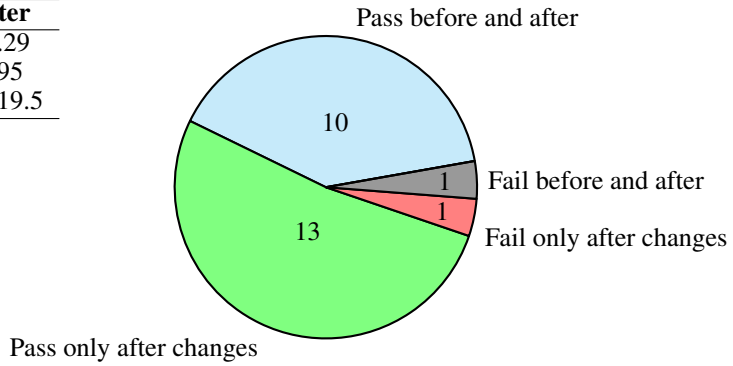
5 Results and discussion

5.1 Results

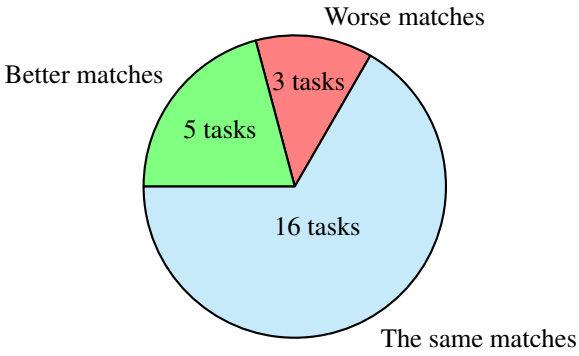
For the tests run on the entire benchmark, 37 tasks are correctly solved initially and 36 are correctly solved after. One additional task is solved, while two are no longer correctly solved. The average times needed and the average number of searched transformations can be seen in Table 2.

For the manually chosen tasks, 11 tasks perform the same. An additional 13 tasks pass with the changes, and one fails after. These can also be seen in Figure 8a.

Out of the 25 randomly chosen tasks, 17 perform the same on both versions of the algorithm, 3 have worse results with the changes, and 5 find more correct correspondences. These results can be seen in Figure 8b.



(a) Results for the 25 manually selected tasks.



(b) Results for the 25 randomly selected tasks.

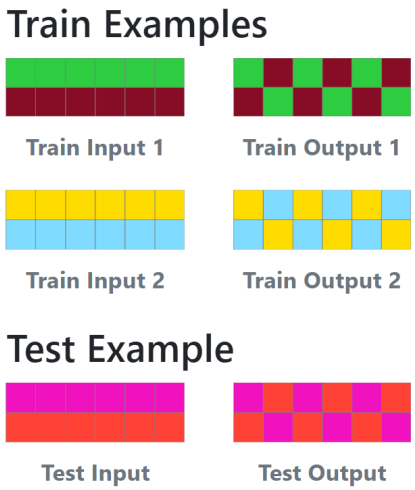


Figure 9: Task e9afcf9a, which fails on the entire algorithm after the changes. Adapted from [8].

5.2 Discussion

Full BEN algorithm on the entire benchmark

The average times for both total solving time and time spent searching transformations are very similar. Accounting for the tests being run only once due to time constraints, the changes made have almost no impact on the efficiency of the algorithm. However, the number of transformations searched decreased by more than 800 on average, which indicates that overall better matches are found in enough cases.

One new task is solved, while two that passed before the changes fail after. One of them times out when searching for correspondences due to too many objects. The training example with the highest number of objects has 24 in input and 24 in output. The other task that fails after the changes can be seen in Figure 9.

This task, e9afcf9a, is always segmented into 2 input objects and 12 output ones, the latter having each one pixel. Before the changes, more matches were done based on colour than on the ranking of the coordinates. After them, all of the top objects in the output were always matched with the top object in the input, with the same behaviour for the bot-

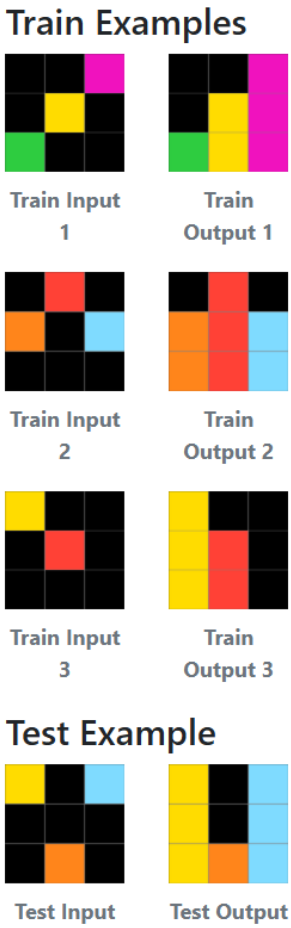


Figure 10: Task d037b0a7, which passes on the entire algorithm after the changes. Adapted from [8].

tom objects. This led to more transformations being found that change the colour instead of keeping it, overfitting on the training examples. Since the colours in the test are not present in any of the train examples, the concept learned uses colour changes that are not correct.

There is also a task that was additionally solved with the changes made, which can be seen in Figure 10. This happens to be one of the tasks included in the randomly selected ones and, by the metrics used there, it performs the same. However, after the changes, the matches found are more restrictive. With the initial implementation, each output object was matched to the correct input object with the highest score, and to an incorrect one with a lower score. With the modifications, only one of the three items is potentially matched with different objects, while the remaining items are only matched with one input object due to increased structural differences.

Alignment in isolation on manually encoded tasks

The results of the manually encoded tasks indicate that the matches are found to be better with the changes made. On both samples, the changes improve the results of the experiments.

Firstly, from the manually chosen tasks, 13 additional matches are found completely correctly after implementing the changes. However, in one of the tasks that was initially matched correctly, the correspondences are no longer correctly identified. A visualisation for that task, 6150a2bd, can be seen in Figure 11.

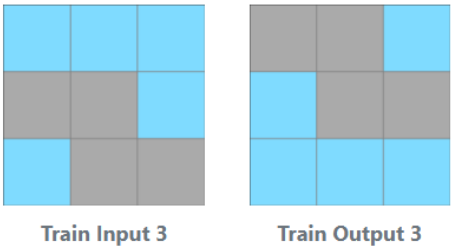


Figure 11: Visualization of ARC task 6150a2bd, which fails on the manually encoded tasks after implementing changes. Adapted from [8].

Nevertheless, this does not significantly impact the overall BEN algorithm. The task is successfully solved both before and after the changes made to the Align part, even though the matches found are worse based on the manual encoding. Initially, the task was solved in around 26 seconds, while with the changes it is solved in around 28 seconds. Although more time is needed to solve it, the difference is not very remarkable.

The results on the randomly chosen tasks indicate a slightly better performance, with a couple more tasks overall having better correspondences after the changes. The five tasks where better matches are found are similar to some of the manually chosen examples. More interesting are the three tasks that have worse matches found. They can be seen in Figures 12, 13, and 14.

None of these tasks are solved by BEN either before or after the changes done in Alignment. The more intriguing part about them is which correspondences are no longer found and why it happens.

In the task in Figure 12 all correspondences are correctly found initially. With the changes the match between the yellow objects is no longer found. This is likely due to the addition of ranked coordinates to their reversed ranking. This is one of the examples where it is not needed, but their existence gives enough structural consistency with the wrong object to get incorrect results. The relative similarity in the structure of all objects becomes high enough that it gets harder to predict what the engine will end up matching. Moreover, a weakness of how encoding is currently done is visible with this example: the values in the ranking need to match exactly to increase structural consistency of the object representation. The current implementation has no way to infer that two objects are still next to each other in the ranking, but in a different position.

The task in Figure 13 is more clear and straightforward. Initially, all correspondences are correctly found, while after the changes the one between the red objects is no longer detected. Most likely, the reason for this is that the ranking of coordinates is weighted more than other features. Since the bounding box of the red object has the same top left corner as the green object it is closest to, they end up having the

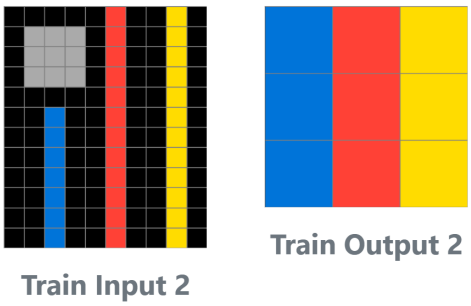


Figure 12: Task 8e1813be, which finds worse correspondences after changes. Adapted from [8].

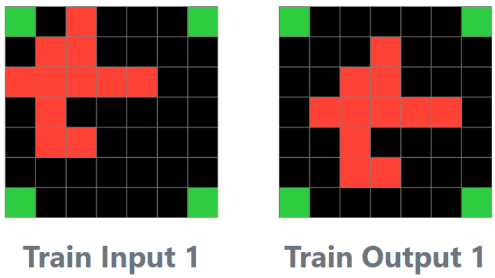


Figure 13: Task a1570a43, which finds worse correspondences after changes. Adapted from [8].

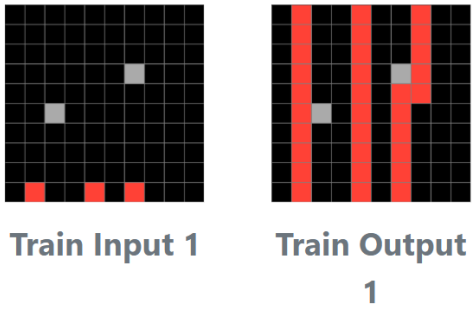


Figure 14: Task d9f24cd1, which finds worse correspondences after changes. Adapted from [8].

same rank in the input and different ones in the output. This could be solved by using the center instead of the top left corner for ranking, but this change could negatively impact other tasks and might be overfitting on the sample. Therefore, we decided to keep it as is.

Lastly, the task in Figure 14 is one where the trade-offs between the initial and changed versions of Alignment are quite visible. In the initial implementation, the two red leftmost objects and the two gray objects from the input are correctly matched to their counterparts in the output. With the changes, all red objects are correctly matched, but the gray ones are not. It happened that in this task, there are more correspondences where the width is the same than correspondences where relational position is more important.

6 Conclusions and Future Work

In this paper, we investigated how to improve the Align part of BEN for solving the ARC-AGI-1 benchmark by finding better matches between input and output objects of the tasks. Considering that the order in which correspondences are found directly affects the first transformations synthesized after, improving alignment has the potential to increase efficiency.

We explored two main approaches to improving the matches found. Firstly, additional features were introduced to better represent spatial relations between the objects, such as ranked coordinates. Moreover, two features were added to better encode how much similarity there is between the colours of objects. Secondly, a heuristic weighting strategy was introduced to adjust the importance of features depending on characteristics of the tasks. The two particular cases taken into account were having the same number of objects in the input and output images, as well as having grids of different sizes between the input and output.

The results indicate that these changes improve correspondence detection overall. When evaluating Alignment in isolation, the modified version achieved substantially better performance on manually selected tasks and showed improvements on randomly selected tasks while introducing limited declines. These results suggest that putting more weight on relational information and adapting the importance of attributes can reflect better how matches appear in ARC tasks.

At the same time, the experiments indicate that it is hard for a single feature configuration to capture how matches are made across all tasks. All the different designs we tried offered some exchanges. Some failures happened because of the newly introduced spatial relations creating structural similarities that were not intended. Others are caused by overemphasizing certain features with the weighting heuristic when they should not be. These trade-offs just highlight how diverse the ARC tasks are and how complex it is to try and capture the important features and relations across the entire benchmark.

Future work can focus in many different areas. One direction can be that of spatial representations. Additional features can be added using alternative coordinate systems, or some richer encodings that can capture more dimensions than just a ranking, such as objects being closest to each other. Colour encoding could also be further explored, as the current solution found is still quite simplistic. Another interesting direction to investigate in the future is matching multiple objects in the input with one object in the output. In some ARC tasks, an output object may use attributes of multiple input ones, and the features necessary to find the correct second match might be completely different. Lastly, the current solution does not encode any features related to patterns or symmetry. Future work could explore how it would be possible to make use of these global attributes.

Overall, this work shows that incorporating richer relational information and task-dependent heuristics can improve object alignment within BEN and provides a foundation for further exploration of analogy-based program synthesis approaches for ARC.

7 Responsible Research

To ensure the scientific integrity of the project, this section discusses the reproducibility of the experiments, the limitations of the experiments conducted, as well as the usage of AI throughout the project.

Reproducibility of experiments

One of the most important considerations of research is ensuring that all experimental findings can be reproduced and thus verified by other researchers. Both the manually selected subset and the randomly selected subset, 25 tasks each, give entirely deterministic results. To facilitate complete reproducibility, the code alongside the JSON files containing the input objects, output objects, and expected matches are available on GitHub. The code is available on the DivideAlignConquer.jl repository, while any additional scripts or JSON files used for evaluation are on the DivideAlignConquerExperiments.jl repository.

The experiments on the entire benchmark with the full algorithm are harder to fully reproduce. While the number of solved tasks and the average number of searched transformations is consistent across runs, the average times might differ slightly. Due to time constraints, these experiments were only run once instead of using an average of multiple repetitions. Although using only average timings across all 400 tasks should alleviate this problem, differences might still be observed in these results.

Limitations of the experiments

While the isolated testing of the Align component shows encouraging improvements, with the manually chosen tasks going from 11 to 23 correctly matched instances, it is still important to put these metrics into the broader context of the entire benchmark.

Firstly, the isolated evaluation was done on a cumulative sample of 50 tasks out of the 400 training tasks the benchmark offers. Although the results obtained on this subset are good, they still only represent a small fraction of the entire ARC public dataset. Therefore, these results cannot be generalized as a performance guarantee across the entire benchmark. We tried to alleviate this by randomly sampling 25 tasks, but the fact remains that the sample is too small in size to ensure better performance overall.

Secondly, the small sample size introduces another limitation related to selection bias. The 25 manually selected tasks are deliberately included to observe performance for the chosen heuristics, and most of them were specifically picked for their suitability of showcasing the improvements the changes bring. To counter this bias, we ensured that tasks where the initial algorithm performed well were also present. However, it is complicated to completely remove bias when manually selecting tasks. For the randomly selected examples, while bias is no longer an issue, some filtering constraints were used to ensure their relevance, such as omitting tasks with only one input object, or tasks that were conceptually unfeasible to segment.

Due to the great diversity of the ARC-AGI-1 benchmark, if another researcher were to select or sample these 50 tasks, they could arrive at different results or reveal alternate trade-offs of the changes chosen.

AI usage

Throughout this project, AI has mostly been used to help with unfamiliar concepts of the new programming language, Julia. LLMs were also used sometimes to find better phrasings for some of the paragraphs in the paper or to provide useful packages for LaTeX. The LLM used was ChatGPT on model GPT-5.5.

References

[1] François Chollet. On the measure of intelligence. *CoRR*, abs/1911.01547, 2019.

[2] J. S. Wind. Arc kaggle competition: 1st place solution. Kaggle, 2020.

[3] A. de Miquel, R. G. Corominas, and Y. Ariyasu. Arc kaggle competition: 2nd place solution. Kaggle, 2020.

[4] Filipe Marinho Rocha, Inês Dutra, and Vítor Santos Costa. Program synthesis using inductive logic programming for the abstraction and reasoning corpus. *arXiv preprint arXiv:2405.06399*, 2024.

[5] Sumit Gulwani and Prateek Jain. Programming by examples: Pl meets ml. In *Proceedings of the Asian Symposium on Programming Languages and Systems (APLAS)*, volume 10695 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2017.

[6] Jonas Witt, Sebastijan Dumančić, Tias Guns, and Claus-Christian Carbon. A divide-align-conquer strategy for program synthesis, 2024.

[7] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1):1–63, 1989.

[8] ARC Visualizations. Arc visualizations, 2024. Accessed: 2026-05-30.

[9] Douglas R. Hofstadter. Analogy as the core of cognition. In Dedre Gentner, Keith J. Holyoak, and Boicho N. Kokinov, editors, *The Analogical Mind: Perspectives from Cognitive Science*, pages 499–538. MIT Press, 2001.

[10] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[11] HERB AI. Herb ai. https://herb-ai.github.io/, 2026. Accessed: 2026-06-02.

[12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.

[13] Sebastijan Dumancic. analogies. https://github.com/sebdumancic/analogies, 2026. GitHub repository, accessed 2026-06-21.

[14] Amin Hashemi and Elisabet Tubau. Global relations versus object relations in visual analogies. *Thinking & Reasoning*, 31(1):109–135, 2025.

[15] Mauro Manassi, Yuki Murai, and David Whitney. Serial dependence in visual perception: A meta-analysis and review. 23(8):18, 2023.

[16] Radoslaw M. Cichy, Nikolaus Kriegeskorte, Kamila M. Jozwik, Jasper J.F. van den Bosch, and Ian Charest. The spatiotemporal neural dynamics underlying perceived similarity for real-world objects. *NeuroImage*, 194:12–24, 2019.

A Additional figures

A.1 Example for analogical reasoning

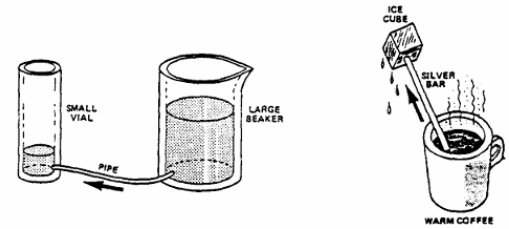


Figure 15: Additional vizualization of water flow and heat flow analogy example, taken from [7].

A.2 Tasks skipped from randomly selected ones

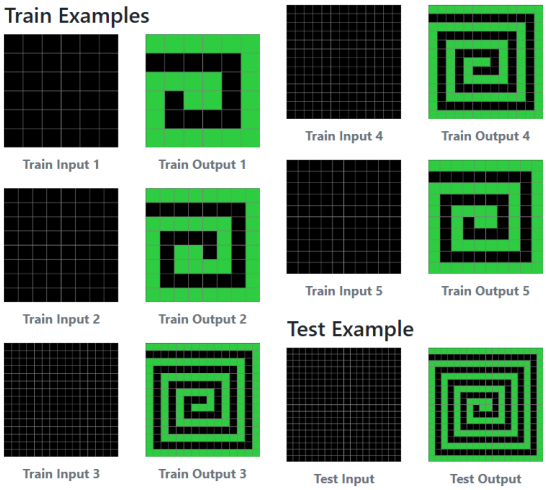


Figure 16: Task 328e73c20, adapted from [8]. Considered not relevant to testing Alignment since there are no input objects.

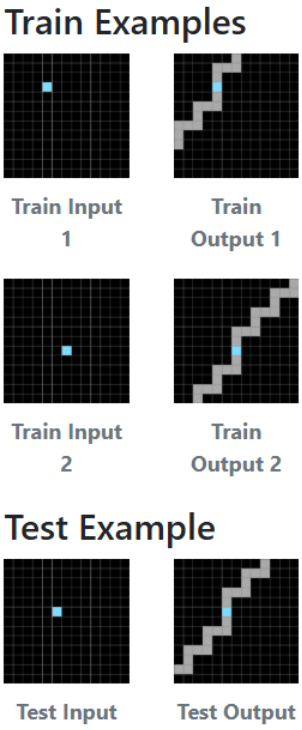


Figure 17: Task 328e73c20, adapted from [8]. Considered not relevant to testing Alignment since there is only one input object.

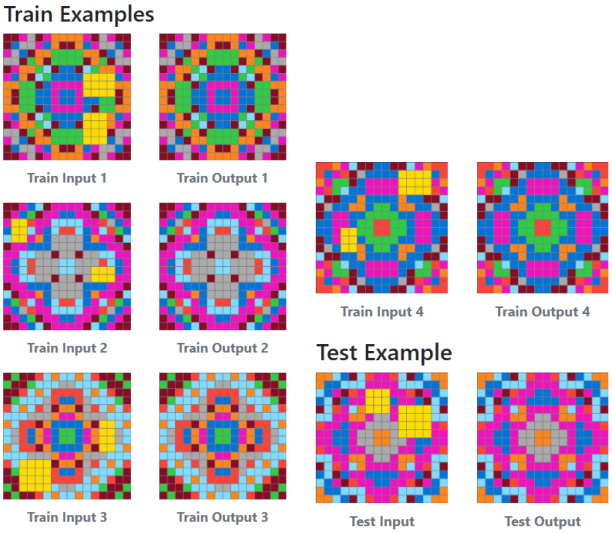


Figure 18: Task b8825c91, adapted from [8]. Considered hard to conceptually segment.