



## A Prolog application for reasoning on maths puzzles with diagrams

Riccardo Buscaroli, Federico Chesani, Giulia Giuliani, Daniela Loreti & Paola Mello

To cite this article: Riccardo Buscaroli, Federico Chesani, Giulia Giuliani, Daniela Loreti & Paola Mello (2022): A Prolog application for reasoning on maths puzzles with diagrams, Journal of Experimental & Theoretical Artificial Intelligence, DOI: [10.1080/0952813X.2022.2062456](https://doi.org/10.1080/0952813X.2022.2062456)

To link to this article: <https://doi.org/10.1080/0952813X.2022.2062456>



© 2022 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.



Published online: 19 Apr 2022.



Submit your article to this journal [↗](#)



View related articles [↗](#)



View Crossmark data [↗](#)



# A Prolog application for reasoning on maths puzzles with diagrams

Riccardo Buscaroli, Federico Chesani , Giulia Giuliani, Daniela Loreti and Paola Mello

Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

## ABSTRACT

Despite the indisputable progresses of artificial intelligence, some tasks that are rather easy for a human being are still challenging for a machine. An emblematic example is the resolution of mathematical puzzles with diagrams. Sub-symbolical approaches have proven successful in fields like image recognition and natural language processing, but the combination of these techniques into a multimodal approach towards the identification of the puzzle's answer appears to be a matter of reasoning, more suitable for the application of a symbolic technique. In this work, we employ logic programming to perform spatial reasoning on the puzzle's diagram and integrate the deriving knowledge into the solving process. Analysing the resolution strategies required by the puzzles of an international competition for humans, we draw the design principles of a Prolog reasoning library, which interacts with image processing software to formulate the puzzle's constraints. The library integrates the knowledge from different sources, and relies on the Prolog inference engine to provide the answer. This work can be considered as a first step towards the ambitious goal of a machine autonomously solving a problem in a generic context starting from its textual-graphical presentation. An ability that can help potentially every human-machine interaction.

## ARTICLE HISTORY

Received 8 October 2020

Accepted 30 March 2022

## KEYWORDS

Multimodal solver; logic programming; mathematical puzzles; spatial reasoning

## Introduction

Over the last years, Artificial Intelligence (AI) achieved results that were just unthinkable some decades ago. Sub-symbolical techniques were successfully applied to a plethora of contexts, like image recognition, Natural Language Processing (NLP) and many aspects of the problem-solving process, often achieving performance that equal or overcome the abilities of a human being. Nonetheless, there still exist some tasks that are rather easy for a person but extremely complex for a machine. From the point of view of cognitive science, these tasks involve different types of intelligence (e.g. mathematical, spatial, logical, or common-sense reasoning), which must be applied and combined in the correct way in order to accomplish the final goal (Schneider & McGrew, 2012). In these tasks, computers cannot overcome human abilities because they still lack effective ways to automatically arrange together various techniques and the knowledge brought by each one. As also pointed out by Chesani et al. (2017), the best method to perform this combination of techniques is unlikely to be fully sub-symbolical, because arranging different intelligences together is not a matter of pure perception, like image processing, or voice recognition. It is rather a *reasoning* operation.

Mathematical puzzles are a perfect example of tasks that are quite easy for humans but still impossible for a machine. Indeed, as described by Chesani et al. (2017), one of the most interesting challenges of modern AI is the following. ‘By the middle of the 21<sup>st</sup> century, (a team of) fully autonomous agent(s) shall win a mathematical puzzle competition against primary school students, winners of the most recent competitions.’

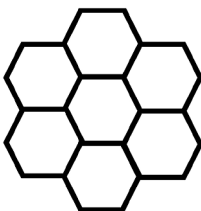
In particular, those puzzles that require the interpretation of both a natural-language text and a diagram pose interesting questions to AI. Some of them can be solved by primary-school children, but no computer can autonomously provide a solution when only the textual and visual descriptions are given as input.

For example, consider the problem in Figure 1, assigned during an international competition between primary school students organised by PRISTEM Research Centre.<sup>1</sup> The puzzle requires to solve a simple map-colouring optimisation problem, which would be trivial for a computer, if only text and diagram were converted into a machine-oriented formulation of the problem beforehand.

Obviously, the key challenge for computers is not connected to the complexity of the calculation required – which is quite limited – but rather to the steps that should precede the actual resolution, i.e. (i) understanding the diagram; (ii) understanding the text; (iii) identify the best modelling and solving technique; (iv) possibly identify hidden knowledge. Moreover, these steps would not be necessarily performed in the mentioned order (Chesani et al., 2017). Remarkably, the gaps of NLP techniques in understanding the data and the question of the problem from the text cannot account alone for the machine’s incapacities, because all sub-tasks must be investigated in a comprehensive *multimodal* approach. Sub-symbolical approaches are crucial to effectively grasp the content of image and text, but the information derived in this way must be processed altogether and – if need be – combined with common-sense knowledge. This assembly task is akin to what is required by Visual Common-sense Reasoning on an image (Zellers et al., 2019): in order to grasp those contents that go beyond the recognition-level perception (i.e. beyond the ‘simple’ detection of objects and their attributes), a kind of cognition-level reasoning is required (e.g. to deduce the intents of the subjects in a scene). Analogously, a tool combining the textual and graphical description of a maths puzzle must be capable of – at least – inferring conclusions from given premises and rules. In our opinion, a suitable tool for this task is a *logic-based* framework.

Mathematical puzzles with diagrams are an excellent training scenario to explore how logic can act as glue between different symbolical and sub-symbolical techniques to properly realise a multi-modal approach. The things we can understand from this field are much likely to be useful in more complex scenarios involving human–computer interaction like for example, the interpretation of textbooks and manuals devoted to query answering or automatic problem solving.

In this work, we focus on the challenging scenario of maths puzzles with diagrams, and we propose a Prolog library that interacts with image-processing software to extract information from the problem’s graphical representation. The library allows spatial analysis at different levels of complexity, from points to more elaborated puzzle-specific shapes, enabling the machine’s reasoning towards the identification of the puzzle’s solution.



*If you had to colour the figure in such a way that no two adjacent hexagons have the same colour, how many different colours would you need?*

Figure 1. Example of maths puzzle with diagram. Translated from the Italian version available online<sup>1</sup>.

Our proposal does not aim to fill all the gaps in autonomous puzzle resolution. In particular, it does not encompass techniques to discover hidden knowledge or understand the text, but assuming to have a convenient method to extract the relevant information from text, the library is able to integrate such a knowledge with the one extracted from the puzzle's diagram. As a first training ground, the library was applied to a subset of the maths puzzles belonging to the PRISTEM Research Centre catalogue<sup>1</sup> of Bocconi University of Milan, Italy.

This study fills some of the gaps and underlines the shape of the obstacles that still stand in the way of the ambitious goal of a machine autonomously solving maths puzzles from their textual-graphical descriptions.

The contribution of this work can be summarised as follows:

- A brief analysis and classification of the math puzzles composing the considered case study in order to underline their most relevant characteristics from the solver point of view.
- The definition of the framework architecture of a system able to integrate the various modules that are necessary to address the different tasks required by puzzle resolution.
- A three-layer multimodal Prolog library devoted to puzzle resolution, addressing low-level diagram processing, high-level spatial reasoning and problem solving. The library also makes use of a Prolog notation specifically conceived to simplify the elicitation of the puzzle's constraints, and express its query in a clear and concise way.
- An experimental evaluation of the library's performance when solving different problems, coupled with a qualitative study of the framework's strength and weaknesses.
- The open source code of the library, as well as an online demo to verify its working principles through a web interface.

## Related work

Games have always been an important training ground for AI. The first remarkable result was obtained when DeepBlue from IBM (Campbell, Jr., & Hsu, 2002) was able to beat the chess champion Kasparov in 1996. More recently, AlphaGo from Google DeepMind defeated Fan Hui, one of the best players of Go in the world (Silver et al., 2016).

Despite these successes, AI still lacks the ability to defeat a fourth grade student on a simple maths or logical puzzle provided in natural language text and/or graphical form. Several works addressed the resolution of maths world problems by means of a declarative logic-based language, like Prolog. This solution is particularly suitable for those logic puzzles – like the ones collected in (Smullyan, 1978) – aiming at determining the truth value of a proposition, given some assertions known to be true (Alzboon & Nagy, 2020; Nagy, 2003; Ohlbach & Schmidt-Schauß, 1985; Ramsey, 1986).

Csenki (2006) focuses on a specific kind of puzzles connected to the numbers-in-diagram class, and proposes a logic-based solution employing a combination of discrete mathematics and Prolog. However, the approach was only devoted to exploit the power of first order logic to create an effective declarative solver, rather than an autonomous, multimodal one. In (Lev et al., 2004), the authors propose a solver for logical puzzles expressed in an intermediate language. An automatic translation into first-order logic enable the resolution through a theorem prover. More recently, Dries et al. (2017) adopt a two-step approach to solve probability word problems. In the first step, the text in natural language is analysed and converted into an ad-hoc declarative modelling language, which takes inspiration from the entities defined in (Hosseini et al., 2014). Then, in the second step, Probabilistic Inductive Logic Programming (PILP; De Raedt et al., 2008; De Raedt & Kimmig, 2015) is used to infer the solution. Akin to Statistical Relational Learning (SRL; Koller et al., 2007; De Raedt et al., 2016), PILP is particularly suitable for the domain of probability world problems because it allows to manage the complexity through

logic, while uncertainty is addressed by probabilistic reasoning. A combination of logical inference and statistical approach is used also in (Liang et al., 2016) to transform the text of maths world problems into an intermediate form, and then into a first-order logic program.

Answer Set Programming (ASP; Baral, 2009; Gelfond & Lifschitz, 1991) is employed by Mitra and Baral (2015) to solve 'logical grid puzzles', intended as maths problems expressed in natural language without diagrams, whose solution can be derived by building a table with the input data. The NLP task of converting each textual constraint of the problem into an ASP predicate is carried out by a Maximum Entropy classifier (Berger et al., 1996). Logical grid puzzles are also the subject of (Cheung, 2016b), who focuses on the extraction of Prolog clauses instead of ASP constraints.

Similarly to our work, the goal of all these approaches is to make a step towards the definition of an autonomous puzzle solver. The main difference is in the type of problems considered, because none of the approaches described so far addresses the case of maths puzzles with diagrams.

The extraction of precious information from the puzzle's diagram is a delicate operation which – we believe – cannot be carried out with the adoption of sub-symbolical techniques only. Spatial reasoning and logic are essential to correctly identify all important data in the figure. A logical expression is the output of the tool described in the work by Seo et al. (2015), which was devoted to the resolution of multiple-choice geometry questions from high-school tests expressed in both natural language and diagrams. Cheung (2016a) gives a description of spatial reasoning puzzles, highlighting that the truth of some logical constraints depends on spatial relation which may or may not be described in the texts, but is certainly shown in diagram. The solving procedure is enabled by a manual conversion of the figure into Prolog atoms and terms.

Physics questions with texts and diagrams are the subject of the work by Forbus et al. (2009), which employ analogy reasoning to create a system that is agnostic of the puzzle's context in the beginning, but can create a parallelism between the given problem and a set of previously given solved examples.

Other related works (Bhatt & Flanagan, 2010; Loreti et al., 2019) focus on spatiotemporal reasoning in specific contexts. Bhatt and Flanagan (2010) in particular, apply abduction to automatic cinematography in order to control the cameras and the perspectives given a graphical and textual description of the scene to be constructed.

Strongly connected to this work are the multimodal approaches in the field of Visual Question Answering (VQA), which aim to provide a natural language answer to an image-related question. Most of the works (Antol et al., 2015; Fukui et al., 2016; Krishnamurthy et al., 2017; Lu et al., 2016) on VQA have a strong sub-symbolical nature and mainly involve neural networks: the possible answers to a question are seen as categories, and the network can be trained to learn the mapping between the features of text and image, and the correct answer. Notably, the work by Andreas, Rohrbach, Darrell and Klein (2015) combines this strategy with a semantic analysis of the question. Being devoted to answer different kinds of questions, the system includes different neural networks in a modular way. The structure of the sentence and its words are used to guide the system through the right combination of modules to compose the neural network more suitable for the case.

The work by Kembhavi et al. (2017) focuses on the interpretation of text and diagrams from primary school tests with a multimodal approach. The authors evaluate state-of-the-art VQA approaches on a 'textbook' dataset of over 1,000 lessons and 26,000 multimodal questions, highlighting their strengths but also their poor performance on some tasks. For example, when the text shows a strong lexical variation w.r.t. the lesson, when the size of the network generated from the input lesson is so big that making it fit into a single memory becomes a non-trivial task, or – more interestingly from our point of view – when the question cannot be answered by simple lookup but requires reasoning.

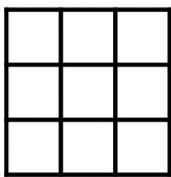
## The mathematical puzzles case study

The PRISTEM catalogue<sup>1</sup> is an archive of maths puzzles assigned to the participants of the ‘Giochi d’Autunno’ (literally ‘Autumn Games’) competition organised every year by the Bocconi University since 1994. The competition encompasses a sequence of matches from national to international level involving children of 4<sup>th</sup> and 5<sup>th</sup> grade of primary school. Hence, the given puzzles are not very complex and do not require advanced mathematical or logical skills. They can be abstract or maths-world problems and their answer can be a word, a number, or a graphical sketch. Some puzzles have more than one solution, but the participants are rarely required to provide them all. According to the Cattell-Horn-Carroll theory (Schneider & McGrew, 2012), the human skills required to solve these puzzles are structured into *strata*, where the lower layers identify very simple concepts, which are put together and elaborated by the layers above.

A considerable portion of the catalogue is composed of puzzles with a diagram. In particular, 60% of puzzles with figure are ascribable to the following categories (Gambetti et al., 2020).

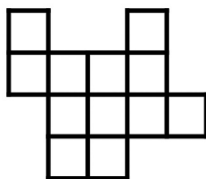
- *Geometrical figures*; which usually require the identification and counting of polygons inside the diagram. The text explicates the kind of polygon and, possibly, some constraints. An example of these puzzles is given in Figure 2a
- *Spatial logic*; such as tiling puzzles, or games where a grid must be partitioned into a certain number of equal pieces (see the example in Figure 2b). Several variants are possible, where pieces can be rotated and/or overturned.
- *Numbers-in-diagram*; i.e., puzzles that require inserting numbers into the boxes of a diagram according to a certain logic explained in the text. For some puzzles, the proposed diagram has some boxes already filled with a number, which must be recognised to further constrain the solution. Examples of this category are Figures 1, 2c and 2d.

From the image processing point of view, the task of extracting relevant information from these diagrams can be seen as a VQA: the puzzle’s input data contained in the figure are retrieved through a set of queries to the image processing engine. This obviously entails the non-trivial ability to formulate the right image-related questions, which are helpful to complete the input. What is most relevant for the solver implementation is that some of these puzzles (around 18%, often ascribable to the category of geometrical figures) require only the image analysis and some basic concepts of geometry, whereas a significant part can be interpreted as a Constraint Satisfaction Problem (CSP; Russell & Norvig, 2010).



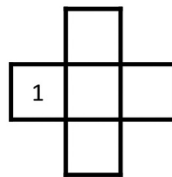
How many squares of 4 pieces are there in the figure?

(a)



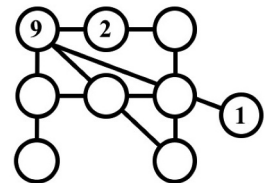
The figure is composed of two identical pieces. Retrace their edges. Pieces can be rotated and overturned.

(b)



Put the digits from 2 to 5 into the white boxes. The sum of the digits on the horizontal line must be equal to the sum of those on the vertical line.

(c)



Put the numbers from 3 to 8 in the empty boxes. The sum of the numbers on the same line must be 18. What is written on the left lower box?

(d)

**Figure 2.** Other maths puzzles with diagram from PRISTEM Research Centre catalogue<sup>1</sup> representing examples of the three categories: *numbers in a diagram* (Figure 2c and 2d), *geometrical figures* (Figure 2a) and *spatial logic* (Figure 2b).

CSP can be modelled through a set  $\{X_1, \dots, X_n\}$  of variables, a set  $\{D_1, \dots, D_n\}$  of domains of each variable, and a set of constraints  $\{C_1, \dots, C_m\}$ , which may involve one or more variables thus limiting the values they can be assigned.

As an example, consider the puzzle of [Figure 2c](#). Each empty cell of the grid can be seen as a variable, to which we can assign a value from 2 to 5. The cell already containing a digit is a variable bounded to the value 1. Therefore, the puzzle can be formalised through a CSP with variables  $\{X_1, \dots, X_5\}$  and domains  $D_1 = \{1\}$ ,  $D_2 = D_3 = D_4 = D_5 = [2, 5]$ . The requirement ‘the sum of the digits on the horizontal line must be equal to the sum of those on the vertical line’ can be expressed through the constraint  $X_1 + X_2 + X_3 = X_2 + X_4 + X_5$ . Furthermore, the puzzle implies that all the values in the cells must be different to one another, i.e.,  $X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_4, X_1 \neq X_5, X_2 \neq X_3, X_2 \neq X_4, X_2 \neq X_5, X_3 \neq X_4, X_3 \neq X_5, X_4 \neq X_5$ .

The approach to CSP resolution is a search of the solution inside a space of possible results: it starts by performing an initial assignment of each variable to one of the values in its domain. When the process is able to assign all the variables without violating any constraint, a solution of the problem is found. In the case of the puzzle of [Figure 2c](#), there is more than one solution satisfying all the constraints:

$$X_1 = 1, X_2 = 3, X_3 = 5, X_4 = 2, X_5 = 4$$

$$X_1 = 1, X_2 = 3, X_3 = 5, X_4 = 4, X_5 = 2$$

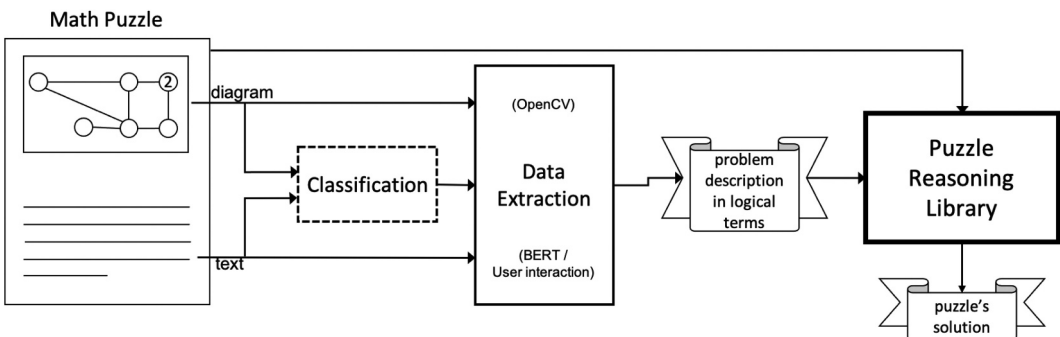
$$X_1 = 1, X_2 = 5, X_3 = 4, X_4 = 2, X_5 = 3$$

$$X_1 = 1, X_2 = 5, X_3 = 4, X_4 = 3, X_5 = 2$$

## Framework architecture

The system we envisage takes as input a textual and a graphical description of the puzzle. Our framework is depicted in [Figure 3](#).

We imagine the input text and diagram processed by a dedicated Classification module, able to classify the type of the puzzle. The implementation of this component is a rather challenging task, which – in our vision – encompasses the ability to extract information through text and image analysis. To this end, this module should be able to perform a *multi-modal* analysis of the various input data.



**Figure 3.** Framework architecture of the puzzle solver.



Then, in order to extract the information needed to build a computer-understandable representation of the puzzle, the system analyses the diagram and the text, together with the puzzle type. In [Figure 3](#) this data extraction is performed by a component that exploits two different technologies dealing with different input formats. Part of the component is devoted to tackle the data within the diagram, while the other part is focused on the NLP task over the puzzle's text. Notice that the Data Extraction module is guided by the puzzle's type into the search of interesting data. Once data is extracted from diagram and text, the system constructs a logic-based (and Prolog-based, in particular) representation of each problem in a defined syntax.

The final step involves the Puzzle Reasoning Library, which takes as inputs the aforementioned representation and the diagram. If need be, the latter is inspected from different points of view, e.g., the reasoning component searches for the presence of polygons, or the possibility to divide the current diagram into sub-shapes. The solution is computed by exploiting the puzzle description in logical terms and the diagram, when needed.

The output of the whole system is the puzzle's result, which can assume the form of a number, a word, or a graphical sketch, often drawn on top of the provided image.

### ***Puzzle classification and objective identification***

The Classification module attempts to answer two very close-related questions: firstly, to which category a certain puzzle belongs; and secondly, which is the general objective of the puzzle.

The task of identifying and understanding the general objective of the problem is still a very challenging task for machines. As described in [Section 2](#), albeit NLP and image analysis techniques have done impressive progresses in the last years, no sub-symbolic approach alone is currently able to tackle this problem, to the best of our knowledge. We might notice, however, that the general objective is usually directly related to the puzzle's type: correctly classifying the puzzle would provide then a strong hint about the right general objective. For example, once it is known that the puzzle belongs to the *number-in-diagrams* category, it is straightforward that the final goal will be to determine the values of a set of variables subjected to some constraints.

Regarding the identification of the puzzle's type, we might notice that both the text and the diagram play a fundamental role in the recognition task. Considering the diagrams only, for example, could lead to ambiguities. It might be that the same diagram can be referred to different puzzles types; e.g. if we focus on the diagrams of [Figure 2b](#) and [2a](#), we might not be able to discriminate the puzzle type, since both the diagrams might be classified as a *geometrical figures* or a *spatial logic* problem. As an interesting side effect of such ambiguity, it happens that the diagram of [Figure 2b](#) can be paired with [Figure 2 a](#)'s text, thus leading to a new maths puzzle.

A similar issue arises if we consider the text in isolation: the identification of the right puzzle type might lead to ambiguities. Indeed, it is the conjunction of text and diagram that usually leads to the proper identification of the puzzle type.<sup>2</sup>

In [Figure 3](#) the classification component is depicted with dashed edges. This is because we still consider that part of our framework as an open issue. We achieved very good results by exploiting a rather specific approach where simple NLP analysis is performed on the text (through Stanford CoreNLP<sup>3</sup> Manning et al. (2014), and WordNet<sup>4</sup>). Indeed, given their primary aim of being recreational for humans, puzzle's texts adopt a very limited vocabulary: as a consequence, solutions that looks for specific terms apparently provide very good results. However, the generality and adaptability of such solutions towards novel or unforeseen problem descriptions is highly questionable.



## Extraction of puzzle's data

Extracting data from the diagram and from the textual description of the puzzle is a difficult task as well. A first consideration is related to the extraction of the *relevant* data: for example, the structure of a sentence, or even some numbers in the text, might not be relevant at all for a certain puzzle. Hence, we focus on implementing Data Extraction Modules that extract the *relevant* information, rather than pursuing an accurate comprehension of every bit of information from the puzzle's description.

A second consideration stems from the fact that knowing in advance the puzzle's type might guide the search for relevant data. For example, if a puzzle is known to belong to the *geometrical figures* class, an immediate question is 'which figure type should we look for in the diagram?', whose possible answers might be 'triangles', 'squares', 'hexagons', etc. In other words, the puzzle type suggests which are the information we should look for.

Implementing such approach requires two steps. Firstly, it is necessary to analyse all the problems and precisely identify which parameters are actually relevant to solve them. This information can be organised into a sort of decision tree, which – starting from general questions, such as providing the class of the problem – deepens the knowledge about the problem's request until the identification of all the necessary data. Secondly, a completely autonomous Data Extraction module would need the ability to follow the decision tree, answering every question from the root to the leaves by means of text and diagram analysis.

In this work, we mainly focus on the first of these steps, whereas for the second step we are investigating two alternative – possibly complementary – approaches: on one side we are exploiting the use of BERT (Devlin et al., 2019), a renowned tool for NLP understanding and query answering. On the other side we are experimenting a query-answering system involving the final user, who is therefore requested to reply to every question following the path of the decision tree. Based on these answers (collected either from BERT or from the user), the system dynamically composes the problem description in logical terms.

As regards the first category of puzzles, i.e. geometrical figures, they require to count specific polygons. Therefore, the parameters that must be provided to the solver involve the kind of polygon and, if need be, some constraint on its characteristics (e.g. its area expressed in number of smaller pieces as in [Figure 2a](#)).

The second category, spatial logic puzzles, can be either jigsaws or games where a grid must be partitioned into a certain number of equal pieces. In both the cases, the possibility to rotate and overturn the pieces represent a parameter of the problem. As regards the jigsaws, the information regarding which are the pieces and which is the grid (as well as the presence of tiles already in place) is not reported in the text, but must be extracted from the diagram. Therefore, this information should not be requested to the user, but autonomously found by the Puzzle Reasoning Library module through image analysis. Similarly, the grid-partitioning puzzles require just the list of actions that can be performed on the pieces, as well as the number of same-shape tiles that must be employed.

Finally, numbers-in-diagram puzzles are characterised by the need to insert numbers into the boxes of a diagram according to a certain logic. The parameters that must be provided are:

- a list of constraints of various natures (expressing the domain in which the numbers must be picked, and the relation that must hold between the digits in certain boxes);
- an indication about the form of the wanted result, which can be either drawn on the diagram or provided through a numeric answer (e.g., "What is written on the left lower box?").

As regards the number's domain, it can be either explicitly elicited (e.g. 'numbers 25, 29, 37, etc. '), or expressed through a starting and an ending number (e.g. 'numbers from 2 to 5'). It can also be parametric as it is for example, in any map-colouring puzzle like the one in [Figure 1](#). There, the implicit domain of the variables is  $[1, N]$  (and we need to minimise the  $N$ ).

About the relation between the numbers in certain boxes, it can be some simple requirement like providing all different values or ensuring adjacent boxes have different values; or it can be some more complicated constraint involving the relative or absolute position of the boxes and some operation on the numbers inside it (e.g. as in the puzzle of Figure 2c, whose request is ‘the sum of the digits on the horizontal line must be equal to the sum of those on the vertical line’).

It is important to underline that for some puzzles the diagram already contains numbers in some boxes, which must be recognised and included in the parameters. Nonetheless, in our vision of the resolution process the tasks of understanding *if* numbers are present and *which* are they are demanded to the Puzzle Reasoning Library, that deepens the analysis of the diagram with the Optical Character Recognition (OCR) feature.

The combination of all this information determines the form of the decision tree. Following such a structure, the system is able to compose the logic-based formulation of the puzzle’s request that feeds the Puzzle Reasoning Library. The current decision tree with the questions is available in the form of an excel file in the GitHub repository<sup>5</sup> of this project.

**The puzzle reasoning library**

The Puzzle Reasoning Library performs image analysis and reasoning on the diagrams driven by the problem description in logical terms that is emitted by the previous Data Extraction module.

From the point of view of diagram analysis, if we consider the examples of puzzles provided in Section 3, it is easy to see that all the problems, regardless to the category they belong, need some low-level features, like the ability to identify points and segments. For this reason, the library provides a first low-level layer to identify the main building blocks of spatial reasoning. Indeed, like the human abilities to solve puzzles, our library – depicted in Figure 4 – is made of overlapping layers.

At the lowest level, the *primitives* module is composed of functions written in C++, making use of the OpenCV<sup>6</sup> library to identify points, segments, shapes, and text/numbers. The choice of OpenCV is motivated by the fact that it is one of the most famous open source libraries for computer vision. Originally written in C++, this library was progressively applied in several fields, from video surveillance to autonomous driving. Thanks to its portability on different operating systems, its interfaces towards various programming languages, and the integration with CUDA and OpenCL to take advantage of graphical hardware, the current implementation has reached outstanding levels of efficiency and performance. OpenCV offers over 2,500 computer vision algorithms for automated

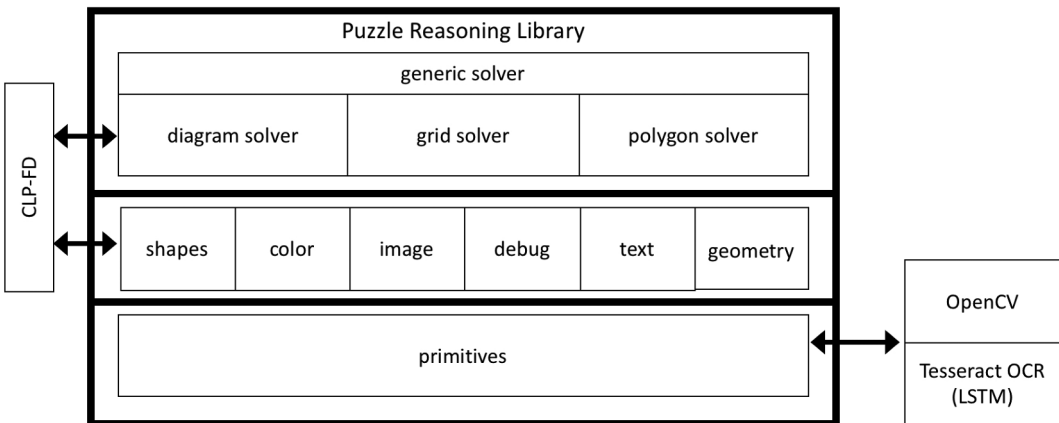


Figure 4. Architecture of the spatial reasoning library.

learning, and a series of additional modules which can be compiled from source to provide advanced or experimental features – e.g. object recognition from images, action recognition or object tracing from video streams, etc.

For the purposes of this project, we used an additional module for scene’s text detection and recognition. In particular, we employed the module which directly interact with Tesseract OCR,<sup>7</sup> a widely-adopted open source software, initially conceived for the digital acquisition of text from scanned documents. Given its initial goal, Tesseract makes a series of hypothesis on the text under analysis. For example, it usually searches for a text spatially organised into lines, with all letters having more or less the same dimension. Those letters that are too small, too big or not aligned with the average of the others are considered as diacritics, punctuation or noise. If, as in the case of puzzle’s diagrams, the text going to be recognised does not fit these initial assumptions, a proper segmentation strategy must be adopted, to subdivide the image into parts where text is searched and recognised. Tesseract offers the possibility to segment the image looking for a single word, a line, or a text with multiple lines, possibly organised into multiple columns. The choice of the segmentation technique must be driven by the kind of spatial organisation of the text that is more often found in the images.

As puzzle’s diagrams may contain text spread all over the figure without a uniform organisation in lines and columns, none of the three segmentation techniques offered by Tesseract is enough to guarantee good recognition performance. Our choice was therefore to repeatedly segment the image as if its content was a text with multiple lines organised into a single column. At each segmentation attempt, we impose to look for text of increasing size. This strategy allows to find letters and numbers scattered across the whole figure (even if they are not aligned), and overcomes the issues deriving from text with different dimensions.

It is nonetheless important to underline that this technique is not sufficient to recognise all possible kinds of text that can be included into a diagram. Letters and numbers that are represented with heavy distortions cannot be identified in this way. Fortunately, as the aim of the puzzles we consider is not to assess the visual abilities of the participant, the number of diagrams with distorted text is very little.

After the segmentation step, which allows to identify portions of the image that are likely to contain text, the recognition process is necessary to assess the actual nature of the found letters and numbers. Tesseract offers the possibility to perform this step by means of a statistical classifier, a neural network-based approach, or a combination of these two. For the considered diagrams we verified that the best results can be obtained with a pure neural network. In particular, we adopted the Tesseract’s strategy that involve a Long Short Term Memory (LSTM) network (Hochreiter & Schmidhuber, 1997) pre-trained on documents of a specific language.

The C++ primitives of the lower level are made available to the Prolog layers above thanks to a C++ interface to SWI-Prolog.<sup>8</sup> By means of this interface, the middle layer extracts and combines together the basic data from the diagram, turning them into more complex information useful to solve the puzzle. This layer includes several modules:

- *geometry*, defines fundamental concepts of plane geometry, such as segments and polygons, based on the most elementary geometrical element: the point;
- *shapes*, contains functions to determine the type of polygons and extract their segments;
- *color*, encompasses colour-related operation, like the identification of the image colour scheme, or the precise colour in a certain point;
- *image*, defines functions to load/release images, and a basic interface to lower-level primitives to draw and cancel objects on the image;
- *debug*, contains basic functions to check the execution flow and show intermediate graphical results of the reasoning process;
- *text*, provides an interface to lower-level primitives to read letters or numbers.

On top of these modules, the third layer is responsible for initiating the solving process by combing the Prolog formalisation of the text with the data that can be extracted from the image. Indeed, the image processing task realises a VQA guided by the puzzle's goal. It is therefore fundamental to establish what the right questions about the diagram are, i.e. the questions whose answers allow to complete the set of puzzle's input data. Considering the categories described in [Section 3](#) it is easy to see that puzzles belonging to the same category require answering very similar questions. For example, finding the correct partitioning of a grid into equal parts, or composing a tiling puzzle with predefined pieces on a grid, both require to know the form and dimension of the grid. On the other hand, puzzles from different categories pose very different questions about the diagram (e.g., geometrical figures puzzles require counting polygons, hence they do not need the concept of grid at all). For this reason, the top layer of the library is composed of a solver module for each considered type of problem.

The *diagram solver* is devoted to the resolution of numbers-in-diagram puzzles. It makes use of the underlying shape and geometry modules to assess the presence of boxes containing numbers and lines connecting those boxes, whereas the actual shape of such boxes (circles, squares, etc.) is obviously irrelevant for solving the puzzle.

The *polygon solver* deals with the recognition of specific polygons in the diagram given some constraint, e.g. on their area or number of edges. To this end, this solver provides functionalities to explore all the possible ways to construct polygons by progressively combining together the diagram's segments.

The *grid solver* deals with spatial logic tasks involving pieces on a grid, and therefore makes use concepts such as the shape and the area of a portion of the figure (irrespectively to the kind of polygon), the possibility to rotate and overturn pieces, and that of filling a part of the grid with a certain piece.

The external interface of the library is realised by the generic *solver*, which loads the puzzle's type and constraints from the Data Extraction module, and dynamically calls the correct solver.

As most puzzles can be interpreted as a CSP, orthogonally to the three layers, the library makes use of Constraint Logic Programming over Finite Domains (CLP(FD)),<sup>9</sup> which extends SWI-Prolog with reasoning over finite domains of integers. Constraint Logic Programming over a theory  $\mathcal{X}$  (CLP( $\mathcal{X}$ ); Jaffar and Maher (1994)) is an extension of Logic Programming that provides the capability of expressing constraints over variables, which are then treated not as 'normal' logic variables, but rather subjected to the constraint theory specified for the type  $\mathcal{X}$ . CLP(FD) allows to consider variables whose range is over a finite domain, and that can be subjected to arithmetic constraints (such as equalities, disequalities and inequalities), but also global constraints such as *all\_different*.

Given these features, the CLP(FD) library seems the best candidate to simplify the declarative elicitation of arithmetic, domain, and combinatorial constraints – involving sets of variable and reification predicates – and perform the problem solving task.

### **Puzzle resolution process**

The resolution process, the extraction of input parameters from the diagram, and the nature of the output we must provide, are all strictly influenced by the puzzle's category. A specific Prolog notation is therefore necessary to specify different puzzle's information (input data and constraints) depending on the problem's type. Our library represents each puzzle through a Prolog term in the form

$$\text{Category}(\text{Parameter1}, \text{Parameter2}, \dots). \quad (1)$$

where the term's name is used to identify the solver of the problem, and the parameters report the input data and constraints derived from the Classification&NLP module.

Parameters expressing constraints must be specified in the form *constr(Constraint)*, where *Constraint* can be either a single or a set of Prolog predicates. In order to simplify the specification of constraints involving collections of objects, the following syntax can be used.

$$\text{constr}(X/\text{Condition} - > \text{Constraint}). \quad (2)$$

The system will apply the *Constraint* to all and only the *X* for which *Condition* is true. Notice that, to the best of our knowledge, the notation proposed in Eq. (2) is new w.r.t. the application domain, but is rather similar to many other notations used to apply constraint on a restricted set of elements, selected from a larger set through the specification of a condition.

The top level solver declares a generic predicate

$$\text{solve}(+\text{Problem}, +\text{Img}, -\text{Result}) \quad (3)$$

where *Problem* contains the puzzle's description through the Prolog term *Category(Parameter1, Parameter2, ...)* presented before, *Img* is the OpenCV object representing the image, and *Result* may be unified with a word, a number, or a modified version of the input *Img* with the solution drawn on it. The actual implementation of the *solve* predicate is assigned to the category-specific solvers, which are examined in the following.

### Numbers diagram resolution

In the *diagram solver*, the puzzle's category and definition can be stated through the Prolog following term:

$$\text{numbers\_diagram}(\text{ConstraintsList}, \text{Result}). \quad (4)$$

where the first parameter (*ConstraintsList*) is a list of constraints on the puzzle's variables and *Result* is another *constr* term that the solver will unify with the solution of the problem. The constraints in the *ConstraintsList* can be expressed by either CLP(FD) or other predicates defined by the library for convenience's sake.

For example, the puzzle of [Figure 2d](#) can be described by the following logic form.

$$\begin{aligned} &\text{numbers\_diagram}([\text{constr}(I/(\text{variable}(I), \text{var\_shape}(I, \text{circle})) - > (\text{var\_value}(I, X), \text{Xin}3..8)), \\ &\text{constr}(L/(\text{line}(L), \text{length}(L, 3)) - > (\text{line\_vals}(L, LX), \text{sum}(LX, \# =, 18))), \\ &\text{constr}(R, (\text{vars\_by\_position}([\text{bottom}, \text{left}], [I\_]), \text{var\_value}(I, R)))). \end{aligned} \quad (5)$$

The first *constr* term in the *ConstraintsList* states that "For any variable *I* such that it is associated to a circle shape in the figure, its value must be constrained in [3, 8]". The second *constr* term expresses the requirement that the sum of the numbers on the same line of boxes must be 18. To state this constraint, the *line* and *line\_vals* predicates are employed to address the aligned boxes and the values inside them, respectively. Finally, the last *constr* term is related to the result *R* and specifies we want to extract the value of the variable located in the bottom left box.

The term in Eq. (5) is passed as first parameter to the *solve* predicate of Eq. (3) in order to trigger the image analysis and the resolution process. In particular, for numbers-in-diagram puzzles, the goal is to insert numbers in the boxes of the diagram according to a certain logic. Therefore the image analysis must first focus on extracting information about the presence and disposition of the boxes in the diagram. This information is represented in the form *diag(BoxList, LineList)*. As the variable's name suggest, *BoxList* is a list of boxes, each represented by a specific *box(Poly, Colour, Txt)* predicate, where the parameters clearly report the type of polygon realising the box, its colour, and its possibly enclosed text. *LineList* is the list of spatial relations between the boxes. Such relations are represented through *line(LineElements, Orientation)* predicates, where *Orientation* specifies the *horizontal*, *vertical*, or *oblique* spatial relation between the set of boxes in *LineElements*.

Whereas these definitions have a rather simple semantics, from the image processing point of view, the identification of what is a box, and the process of distinguish it from other closed forms in the diagram is less straightforward. The human brain usually classifies as boxes all those closed spaces corresponding to geometric figures or characterised by relevant symmetries. For this reason,

we currently identify circles, rectangles and equilateral polygons as boxes, but this criterion can be extended by declaring additional shapes accepted as boxes. This operation can be easily performed by changing of the *true\_box(Box)* predicate .

The search of spatial relations between the boxes have similar issues. In some diagrams, adjacent boxes are polygons with common borders or vertex as in [Figure 2c](#). In some others, the spatial relation is defined by straight, curve, continuous or dashed lines connecting the boxes as shown in [Figure 2d](#). Two different kinds of image analysis are therefore needed. To this end, the library first identifies relations of the first type (polygons with common vertex and borders), then considers all the segments that do not belong to any box and performs the second type of search (boxes connected by segments).

Once the image features are extracted, the CSP model of the problem must be defined. As these puzzles require writing numbers inside the boxes, a variable is associated to each box. If the box already contains a number in the original image, the variable is obviously associated with such number.

The number recognition is the most expensive task of the diagram solving process. When the problem does not actually require to recognise numbers (as it is, e.g. in the puzzle of [Figure 1](#)), that task can be skipped by employing the term *diagram(ConstraintsList,Result)*. The puzzle in [Figure 1](#) can be formulated as follows.

$$\begin{aligned} & \text{diagram}([\text{constr}(I/\text{variable}(I) - > (\text{var\_value}(I, X), \text{count\_vars}(N), X_{in1..N})), \\ & \text{constr}((I1, I2)/(\text{variable}(I1), \text{variable}(I2), I1 = I2, \text{adjacent}(I1, I2)) - > \\ & (\text{var\_value}(I1, X1), \text{var\_value}(I2, X2), X1 \# = X2)), \\ & \text{constr}(\text{minimize\_max})], \\ & \text{constr}(R, (\text{vals}(LX), \text{max\_list}(LX, R)))). \end{aligned}$$

where the first *constr* term states that any variable *I* associated with a box must be an integer in the range between 1 and the total number of employed variables; the second *constr* term enforce to assign different values to those variables that are associated with adjacent boxes; and the third *constr* term minimises the total number of employed variables. Finally, the *Result* parameter is again a *constr* term which states to return the maximum value associated to a variable in the solution. The problem can be formulated in the same way by using *numbers\_diagram* instead of *diagram*, but in that case the image processing task would be triggered before any reasoning to look for numbers in the boxes (in vain).

### Geometrical figures resolution

When the puzzle requires to identify and enumerate certain shapes in the diagram, the *polygon solver* must be called. In this solver, the problem type can be specified by two predicates:

$$\text{count\_polygons}(\text{Shape}) \tag{6}$$

$$\text{count\_polygons}(\text{Shape}, \text{constr}(X, \text{Filter})) \tag{7}$$

where *constr(X,Filter)* allows to specify if some additional constraint must be fulfilled besides the congruence with the specified *Shape*. For example, we could require the polygon to be equilateral or have a certain area (as in [Figure 2a](#)). Since in these puzzles the area is usually defined in terms of the number of base polygons composing a grid (e.g. in [Figure 2a](#) we look for squares composed of 4 pieces), a *min\_area(-Area)* predicate is defined to extract the area of the smaller polygon in figure (i.e., the piece). This polygon can be of any kind, not necessarily a square. Another predicate, *grid\_area(+Poly,-N)*, unifies the *N* variable with the number of minimal polygons needed to cover the area of the input polygon *Poly*. As an example, the logic programming description of [Figure 2a](#) is simply:

$$\text{count\_polygons}(\text{square}, \text{constr}(X, \text{grid\_area}(X, 4))).$$

which instructs the solver to identify and count the number of squares whose area is equal to 4 grid elements.

Despite the short logic-oriented description of the problem, the shape recognition task underneath is not trivial. Instead of looking for all possible polygons in figure, and discard the ones not fulfilling the requirements – a strategy which can be time consuming – our algorithm starts from an image segment, and compose a polygonal chain by progressively adding a segment at a time until the desired number is reached. A line cannot be added if it intersects the chain in a point different from the initial one, and if two segments are aligned, they are obviously considered as a single segment. If the required number of sides is reached without closing the chain, the algorithm backtracks on other image segments. In this way, no time is wasted on searching polygons with more edges than required.

### Grid resolution

In case of spatial logic problems, such as tiling puzzles, or games where a grid must be partitioned into a certain number of equal pieces, the *grid solver* module is employed. Similarly to the polygon solver, for this solver the puzzle's logic description is again rather synthetic. It can be represented by two alternative predicates:

$$\text{puzzle}(\text{Options}) \tag{8}$$

$$\text{divide\_same\_shape}(N, \text{Options}) \tag{9}$$

where the *puzzle* predicate is used to address tiling puzzles (where a set of given pieces must be put on a grid); whereas *divide\_same\_shape* is devoted to situations where only the grid is provided in the figure, and the text requires to divide it into a certain number of equal pieces. The *Options* parameter is provided because often the puzzle allows to perform spatial operations on the pieces, such as *rotate* or *overturn*. The parameter *N* of *divide\_same\_shape* instead expresses the number of pieces in which the grid should be divided.

As an example, the puzzle in [Figure 2b](#) can be described by the following.

$$\text{divide\_same\_shape}(2, [\text{rotate}, \text{overturn}]).$$

In case of *puzzle(Options)* problems, the diagram reports both the grid and the pieces to be scattered on it. So, all the external borders of the picture's elements are considered and the bigger one is found to be the grid (which is reasonable because it has to contain all the others). For both *puzzle* and *divide \_ same \_ shape*, the grid analysis starts by identifying its shape in terms of all its minimal building blocks: the smallest square of the grid, which the solver will represent through the position of its upper left vertex. The grid is therefore rendered as if it was on a Cartesian plane: a predicate *grid\_diag(LX,LY,Grid)* serve the purpose. *LX* and *LY* are the lists of axis coordinates corresponding to the upper left vertices, and *Grid* is a matrix (a list of lists with the same length) where each element is either 0 or -1, to signify that a square was found or not found on the corresponding coordinates. Indeed grids might not always be convex forms, such as squares or rectangles. The strategy of representing the grid through two lists of coordinates and a matrix of 0/-1 allows to easily represent grids with various forms.

The solving strategy searches the first uncovered grid's square and tries to cover the connected space around it with one of the pieces, possibly rotating and overturning it if such options are allowed. If the solver cannot cover the connected space, it backtracks by reconsidering the latest added piece. In case of *divide \_ same \_ shape*, the algorithm does not know the shape of the pieces, but only that their area must be the total area of the grid divided by *N*. So, the algorithm makes a first attempt to define a possible shape and generates *N* identical pieces to occupy the grid.



Since for a machine the challenges in solving one of these puzzles is clearly not in the dimension of the problem, we do not investigate the performance of the library for increasing number of variables and constraints. However, we underline that the most time-consuming task is text detection and recognition, whereas the executions of the grid, polygon or diagram solvers (once the numbers are recognised) all require a very limited amount of time, in the order of milliseconds. This was indeed expected, because the dimensions of the considered problems are extremely small with respect to the computing capabilities of modern hardware.

The whole source code of our spatial reasoning library can be found on GitHub<sup>5</sup>.

## Experimental evaluation

In our framework, the puzzle resolution process involves several different tasks, from the extraction/definition of the logic-based description of the puzzle, to the resolution of the problem specified in a computer-understandable form. The latter in particular is of less interest: as we discussed in [Section 1](#), in the AI research field many successes have been obtained when tackling with specific problems. With this respect, the mathematical puzzles are indeed toy problems w.r.t. the currently available resolution techniques. For example, let us consider the numbers-in-diagram category games: when properly mapped, the puzzles become CSP problems of some ten variables and twenty constraints at most (remember that puzzles are aimed to human recreation). The performance of Constraint Programming solvers is usually measured against problem instances with tenth of thousands of variables, and hundred thousand constraints. As a consequence, the computational time required to solve a properly represented puzzle is usually in the order of nanoseconds, thus making any measurement meaningless. For this reason, in [subsection 5.1](#) we briefly discuss the computational performances of the solving process from the final user perspective, by simply exploiting the demo website.<sup>10</sup>

Of the greatest interest instead is the evaluation of the quality of the solving process. As previously discussed, the ‘hard’ part of solving the puzzles is getting a correct computer-understandable representation of the puzzle. In [subsection 5.2](#) we discuss a more qualitative evaluation of our approach, trying to highlight achievements but also weaknesses of the current framework. We do not introduce any metric on the quality of the obtained puzzle’s model, but rather we investigate if and when our framework manage to achieve a puzzle model that will in turn lead to a correct solution.

### *The demo website and its performance*

A demo of our framework is available online<sup>10</sup>, and interested users can freely experiment with part of the presented tools.

As shown in [Figure 5](#), the web-based application presents four areas: the ‘Human readable’ area is devoted to show the puzzle in its original aspect, while the ‘Prolog translation’ area shows the prolog query whose parameters are indeed the problem description in logical terms. The user is also invited to freely change the parameters and experiment with the reasoning library: tips are provided suggesting parameters that indeed will make the puzzle feasible, thus leading to a solution.

The user can freely choose among fifteen puzzles: upon the selection of a puzzle, its image and textual description are shown, together with the problem translation into a Prolog data structure. By pressing the button ‘Solve’, the selected image with its text, and the Prolog query, are processed to the server, which start a Prolog interpreter.<sup>11</sup> The ‘SWIPL Console’ area reports the output of the Prolog interpreter running on the server, while the “Output image” shows those solutions that are required to be drawn on top of the original image (such as, e.g. the geometrical puzzles). At every pressing of the ‘Solve’ button the time to execute the SWIPL invocation is reported below the ‘SWIPL Console’ area.

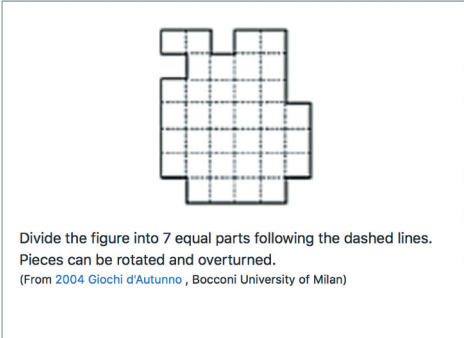
[Games](#)   [Demo](#)   [Download](#)

**Try to...**  
 divide into a different number of pieces, e.g., `divide_same_shape(35, [rotate, overturn])`.

Select a game and press Solve

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

**Human readable:**



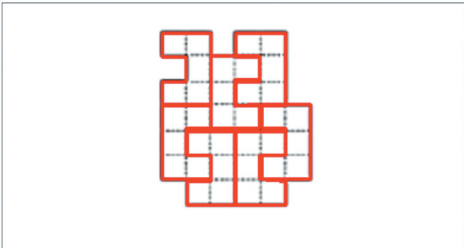
Divide the figure into 7 equal parts following the dashed lines. Pieces can be rotated and overturned.  
 (From 2004 Giochi d'Autunno, Bocconi University of Milan)

**Prolog translation:** ...

```
divide_same_shape(7,[rotate,overturn]).
```

Solve

**Output image:**



**Swipl console:**

```
?- ['load.pl'].
?- solver:demo('in/img/2004A05.png', 'divide_same_shape(7, [rotate,overturn]),', 'out/img/2004A05-33.png').

Solving...
PIText: divide_same_shape(7,[rotate,overturn]).
result(img(2096764))

Swipl called at Mon, 26 Jul 2021 09:14:57 GMT. Total time: 549ms
```

**Figure 5.** Web interface of the demo application.

In [Table 1](#) we report the execution time (in sec.) requested to solve each of the puzzles available in the demo web application. The reported times are not comprehensive of the network overload. For each puzzle, resolution times are taken for ten consecutive tests, and the average time is reported.

**Table 1.** Execution times of the demo web application. Average over ten runs for each puzzle.

Puzzle #	Type	Time (sec.)
1	Spatial Logic	0.713
2	Numbers-in-diagram	4.672
3	Geometrical figure	0.491
4	Numbers-in-diagram	0.830
5	Spatial Logic	0.653
6	Numbers-in-diagram	11.275
7	Numbers-in-diagram	19.518
8	Numbers-in-diagram	31.968
9	Spatial Logic	0.503
10	Spatial Logic	0.504
11	Numbers-in-diagram	17.710
12	Geometrical figure	0.546
13	Spatial Logic	0.500
14	Numbers-in-diagram	25.401
15	Spatial Logic	0.517

The demo web application currently suffers some limits, partly because of security reasons, and partly as a consequence of some limits of our framework (see, [Section 6](#) for a discussion): (i) it is not possible to upload a puzzle: the user can experiment only with the proposed fifteen examples; (ii) the puzzles are already classified into the right problem type; (iii) the problem description in logical terms is already provided, instead of being generated on the fly: the user can however modify it before starting the reasoning.

### **Qualitative assessment of the approach**

Several different aspects have a great influence on the quality of the system outcomes. Regarding the diagram-related part, there is of course the quality of the image: jpeg artefacts or a too much low resolution might lead the OpenCV library to detect false edges, or to miss existing ones. For example, the recognition of polygons is based on the assumption that edges indeed meet each other in some vertex: the presence or absence of some pixel around such vertex might induce the library to false conclusions. Even, the font adopted for indicating some number directly in the diagram might impact the performances of the OCR component. Scanned images in particular might present distortions of the characters, thus leading to recognition mistakes.

Regarding the text-related part, we might stress that the variability in describing the puzzles heavily affects the data extraction module. Such problem is exacerbated from the fact that the available dataset is in Italian language only. We worked on an English-translation of the problems, but we can't exclude that we involuntary introduced some language bias during the translation step. Resorting to translation software lead to poor results, given that the puzzle's texts are usually very short sentences. Alternatively, we investigated the adoption of NLP tools for the Italian language, but when confronted with analogous tools for the English language, the maturity level of the former appears to be lower. A further difficulty relates to the length of the puzzle's texts, combined with (sometime useless) references to a real-life context. While such references help the human to better understand the problem, they might result as misleading information when dealing with NLP tools.

The use of Deep Learning-based approaches might appear as the perfect answer to the problems highlighted above. However, the total number of available mathematical puzzles counts up to around a hundred of examples, a number far insufficient for any Machine Learning training algorithm. Fine tuning of pre-trained neural networks might not be feasible as well, given the small dimension of the training data.

### **Conclusion**

Maths puzzles with diagrams, even the most simple, are a perfect training field for AI. Indeed, despite the impressive progresses made so far, a lot of work is still needed to reach the ambitious goal of a machine autonomously solving a maths puzzle from its original textual and graphical formulation. No sub-symbolical technique can reach this goal alone. The case study of maths puzzles with diagrams inevitably requires a multimodal approach to problem solving, which combines sub-symbolical techniques to extract relevant information from text and diagram into a unique framework. We claim that such task must be guided by reasoning. Logic and its symbolical approach are therefore essential in this scenario.

Focusing on maths puzzles with diagrams, we propose a framework to allow knowledge integration of concepts expressed by the text with what can be extracted from the diagram. In particular, we propose a Data Extraction module, which follows a decision tree of questions from the most general to the most specific in order to deepen the knowledge about the puzzle's request; and a set of primitives, which interact with image processing software to extract and reason upon spatial information about the elements in the figure. Similarly to the human cognitive skills needed to reason on maths puzzles, the functionalities offered by the proposed software are organised into overlapping layers, from the basic to the most complex ones.

The result is interesting not only for the wide set of puzzles that can be solved with our library but, more notably, because it allows to verify that logic programming is the right tool to manage the complexity of the solving process, even when a multimodal approach is needed. Indeed, thanks to Prolog and the CLP(FD) library, we can easily express the knowledge extracted from the image and combine it with a logical description of the natural language text, leaving the burden of finding the best solving strategy for the puzzle to the underlying theorem prover.

Furthermore, we believe that the proposed decision tree of questions useful to data extraction represents itself a relevant contribution of our work, because it can be used by human beings to support the resolution process. For example, it can represent a guide to solve the puzzle for those children experiencing some difficulties in the identification of the correct resolution method; or it can have a didactical value if followed by the children looking for the relevant data in the text – thus helping the teacher to explain the crucial points of the problem description.

Our work also helps to highlight the limits of current approaches and the next steps required. As already discussed, the image processing and OCR software may show some shortcomings when recognising distorted letters and numbers. Besides that, an additional effort is needed to enrich the concepts that the library can manage. For example, the current version of the software can recognise geometrical figures, numbers, letters, and some more general concepts, like for example, the notion of ‘box’, which in the puzzle’s context can be a shape with a variety of forms. However, if the puzzle required to identify different real-world concepts (e.g. arrows, card’s suits, etc.), the integration of other, more sophisticated image recognition techniques into the library would be required.

The spatial reasoning library can manage puzzles that falls into the categories of numbers-in-diagram, geometrical figures, and spatial reasoning. These classes together cover the majority of puzzles from the considered source<sup>1</sup>, but another important part is represented by crypto-arithmetic games. This category could be integrated with a relatively contained effort because most of the spatial reasoning tools required for solving such puzzles is already provided by the library.

In our opinion, the biggest limit of our approach (and also the toughest problem so far) is related to the integration of NLP techniques to properly recognise the puzzle’s type and to convert the problem’s textual description into a set of logic facts and clauses. As mentioned earlier, we investigated different techniques, but none of them provided good results and at the same time proved to be robust enough against unforeseen examples. Other solutions might be inspired by some previous works on this topic (Berger et al., 1996; Mitra & Baral, 2015): we plan to investigate these techniques as well in the near future.

Finally, in the long run a challenging task would be to widen the application scope to contexts other than maths puzzles, where a problem-solving-oriented reasoning is required. An ambitious example of future work is the automated reasoning on technical problems after having understood the content of a related manual (with natural language descriptions and graphical schemas) without human intervention. In this regard, our proposal to integrate sub-symbolical techniques to account for perceptive tasks (like image processing) and symbolical techniques for higher-level reasoning reflects a widespread trend in model AI. In the broad range of scenarios that can benefit from a simplified human–machine interaction, the lesson learned from the multimodal approach to puzzle resolution can be extremely useful.

## Notes

1. Catalogue of maths puzzles – PRISTEM Research Centre, Bocconi University of Milan. <http://matematica.unibocconi.it/articoli/archivio-giochi>
2. Among the so called ‘recreational games’, there are also puzzles whose diagram and text purposely try to fool the user into classifying it in the wrong problem type. In this work we do not consider such a class of puzzles.
3. <https://stanfordnlp.github.io/CoreNLP/>
4. <https://wordnet.princeton.edu/>
5. <https://github.com/ai-unibo/spatial-reasoning>
6. <https://opencv.org>

7. <https://opensource.google/projects/tesseract>
8. <https://www.swi-prolog.org/pldoc/man?section=cplusplus-overview>
9. <https://www.swi-prolog.org/man/clpfd.html>
10. <http://games-ai.disi.unibo.it/games>
11. We choose to rely on the SWI-Prolog Interpreter <https://www.swi-prolog.org/>

## Acknowledgements

This work has been partially supported by the European Union's H2020 projects AI4EU (g.a. 825619) and TAILOR (g.a. 952215).

## Disclosure statement

No potential conflict of interest was reported by the author(s).

## Funding

This work was supported by the Horizon 2020 Framework Programme [825619,952215].

## ORCID

Federico Chesani  <http://orcid.org/0000-0003-1664-9632>

Daniela Loreti  <http://orcid.org/0000-0002-6507-7565>

Paola Mello  <http://orcid.org/0000-0002-5929-8193>

## References

- Alzboon, L., & Nagy, B. (2020). Truth-teller-liar puzzles with self-reference. *Mathematics*, 8(2), 190. <https://doi.org/10.3390/math8020190>
- Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2015). Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799 doi:10.48550/arXiv.1511.02799 .
- Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., & Parikh, D. (2015). VQA: Visual question answering. In *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, (pp. 2425–2433). IEEE Computer Society.
- Baral, C. (2009). *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press.
- Berger, A. L., Pietra, S. D., & Pietra, V. J. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 39–71 <https://aclanthology.org/J96-1002.pdf> .
- Bhatt, M., & Flanagan, G. (2010). Spatio-temporal abduction for scenario and narrative completion. *Proc. of the int. workshop on spatio-temporal dynamics*, Lisbon, Portugal (p. 31–36). IOS Press.
- Campbell, M., Jr., Hoane, A. J. H., & Hsu, F. (2002). Deep blue. *Artificial Intelligence*, 134(1–2), 57–83. [https://doi.org/10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1)
- Chesani, F., Mello, P., & Milano, M. (2017). Solving mathematical puzzles: A challenging competition for AI. *AI Magazine*, 38(3), 83–96. <https://doi.org/10.1609/aimag.v38i3.2736>
- Cheung, B. (2016a). *Spatial reasoning explained*. <http://bennycheung.github.io/spatial-reasoning-explained>
- Cheung, B. (2016b). *Using prolog to solve logic puzzles*. <http://bennycheung.github.io/using-prolog-to-solve-logic-puzzles>
- Csenki, A. (2006). Enigma 1225: Prolog-assisted solution of a puzzle using discrete mathematics. *Computers & Mathematics with Applications*, 52(3–4), 383–400. <https://doi.org/10.1016/j.camwa.2006.03.020>
- De Raedt, L., Frasconi, P., Kersting, K., & Muggleton, S. (Eds.). (2008). *Probabilistic inductive logic programming - theory and applications* (Vol. 4911). Springer.
- De Raedt, L., Kersting, K., Natarajan, S., & Poole, D. (2016). *Statistical relational artificial intelligence: Logic, probability, and computation*. Morgan & Claypool Publishers.
- De Raedt, L., & Kimmig, A. (2015). Probabilistic (logic) programming concepts. *Machine Learning*, 100(1), 5–47. <https://doi.org/10.1007/s10994-015-5494-z>

- Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* Minneapolis, Minnesota (pp. 4171–4186). Association for Computational Linguistics.
- Dries, A., Kimmig, A., Davis, J., Belle, V., & De Raedt, L. (2017). Solving probability problems in natural language. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, Melbourne, Australia, 3981–3987.
- Forbus, K. D., Klenk, M., & Hinrichs, T. R. (2009). Companion cognitive systems: Design goals and lessons learned so far. *IEEE Intelligent Systems*, 24(4), 36–46. <https://doi.org/10.1109/MIS.2009.71>
- Fukui, A., Park, D. H., Yang, D., Rohrbach, A., Darrell, T., & Rohrbach, M. (2016). Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas, (pp. 457–468). The Association for Computational Linguistics.
- Gambetti, E., Buscaroli, R., Chesani, F., Giusberti, F., Loreti, D., & Mello, P. (2020). Artificial intelligence and cognitive psychology: How to solve mathematical problems. *Sistemi Intelligenti*, 32(2), 287–316 doi:10.1422/96329.
- Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4), 365–386. <https://doi.org/10.1007/BF03037169>
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hosseini, M. J., Hajishirzi, H., Etzioni, O., & Kushman, N. (2014). Learning to solve arithmetic word problems with verb categorization. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar (pp. 523–533). Association for Computational Linguistics.
- Jaffar, J., & Maher, M. J. (1994). Constraint logic programming: A survey. *The Journal of Logic Programming*, 19–20(20), 503–581. [https://doi.org/10.1016/0743-1066\(94\)90033-7](https://doi.org/10.1016/0743-1066(94)90033-7)
- Kembhavi, A., Seo, M. J., Schwenk, D., Choi, J., Farhadi, A., & Hajishirzi, H. (2017). Are you smarter than a sixth grader? Textbook question answering for multimodal machine comprehension. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA (pp. 5376–5384). IEEE Computer Society.
- Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., & Pfeffer, A., others. (2007). *Introduction to statistical relational learning*. MIT press.
- Krishnamurthy, J., Dasigi, P., & Gardner, M. (2017). Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 1516–1526). Association for Computational Linguistics.
- Lev, I., MacCartney, B., Manning, C. D., & Levy, R. (2004). Solving logic puzzles: From robust processing to precise semantics. *Proc. of the 2nd workshop on text meaning and interpretation* (pp. 9–16). Association for Computational Linguistics.
- Liang, C., Hsu, K., Huang, C., Li, C., Miao, S., & Su, K. (2016). A tag-based English math word problem solver with understanding, reasoning and explanation. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Demonstrations* (pp. 67–71). The Association for Computational Linguistics.
- Loreti, D., Chesani, F., Mello, P., Roffia, L., Antoniazzi, F., Cinotti, T. S., ... Costanzo, A. (2019). Complex reactive event processing for assisted living: The habitat project case study. *Expert Systems with Applications*, 126, 200–217. <https://doi.org/10.1016/j.eswa.2019.02.025>
- Lu, J., Yang, J., Batra, D., & Parikh, D. (2016). Hierarchical question-image co-attention for visual question answering. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 29, pp. 289–297). Curran Associates, Inc.
- Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. *Association for Computational Linguistics (ACL) System Demonstrations*, Baltimore, Maryland (Association for Computational Linguistics) (pp. 55–60). <http://www.aclweb.org/anthology/P/P14/P14-5010>
- Mitra, A., & Baral, C. (2015). Learning to automatically solve logic grid puzzles. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1023–1033). The Association for Computational Linguistics.
- Nagy, B. (2003). Boolean programming, truth-teller-liar puzzles and related graphs. *Proceedings of the 25th international conference on information technology interfaces, 2003. iti 2003*, Cavtat, Croatia (IEEE). (p. 663–668).
- Ohlbach, H., & Schmidt-Schauß, M. (1985). The lion and the unicorn. *Journal of Automated Reasoning*, 1(3), 327–332. <https://doi.org/10.1007/BF00244274>
- Ramsey, B. D. (1986). The Lion and the Unicorn Met PROLOG. *ACM SIGPLAN Notices*, 21(8), 62–70. <https://doi.org/10.1145/382278.382395>
- Russell, S. J., & Norvig, P. (2010). *Artificial intelligence - A modern approach, third international edition*. Pearson Education.
- Schneider, W., & McGrew, K. (2012). The Cattell-Horn-Carroll model of intelligence. In *Contemporary intellectual assessment: Theories, tests, and issues* (3rd ed.). (pp. 99–144). Guilford Press.

- Seo, M. J., Hajishirzi, H., Farhadi, A., Etzioni, O., & Malcolm, C. (2015). Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1466–1476). The Association for Computational Linguistics.
- Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., & van den Driessche, G., & others. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489. <https://doi.org/10.1038/nature16961>
- Smullyan, R. (1978). *What is the name of this book?* Prentice-Hall, Inc.
- Zellers, R., Bisk, Y., Farhadi, A., & Choi, Y. (2019). From recognition to cognition: Visual commonsense reasoning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 6720–6731). Computer Vision Foundation/IEEE.