Available online at www.sciencedirect.com
SciVerse ScienceDirect
Cognitive Systems Research 22–23 (2013) 35–46
Cognitive Systems RESEARCH
www.elsevier.com/locate/cogsys

An anthropomorphic method for progressive matrix problems

Action editor: Ron Sun

Claes Strannegård [*], Simone Cirillo, Victor Ström

Department of Applied Information Technology, Chalmers University of Technology, Gothenburg, Sweden

Received 30 December 2011; accepted 19 July 2012
Available online 27 August 2012

Abstract

Progressive matrix problems are frequently used in modern IQ tests. In a progressive matrix problem, the task is to identify the missing element that completes the pattern of a pictorial matrix. We present a method for solving progressive matrix problems. The method is not limited to problems that are on the multiple choice format, which makes it potentially useful for solving real-world pattern discovery problems that do not come with predefined answer alternatives. The method is anthropomorphic in the sense that it uses certain problem solving strategies that were reported by high-achieving human solvers. We also describe a computer program implementing this method. The computer program was tested on the sets C, D, and E of Raven's Standard Progressive Matrices test and it produced correct solutions for 28 of the 36 problems considered. This score corresponds roughly to an IQ of 100. Finally, we conclude that it is possible to solve progressive matrix problems without analyzing potential answer alternatives and discuss some implications of this finding.
© 2012 Elsevier B.V. All rights reserved.

Keywords: Anthropomorphism; Cognitive modeling; Inductive reasoning; Raven's Progressive Matrices

1. Introduction

In this paper we describe a method and a computer program for solving progressive matrix problems. Our method is not limited to progressive matrix problems that are on the multiple choice format. Thus it can be used for solving progressive matrix problems both with and without predefined answer alternatives. Problems that are formulated without multiple choice alternatives are generally strictly harder to solve. They are also more relevant to cognitive modeling and more useful for solving real-world problems that do not come with predefined answer alternatives. Our method is anthropomorphic in the sense that we implemented certain problem solving strategies that have been reported by high-achieving human problem solvers.

1.1. Intelligence

In 1912, the *intelligence quotient* (IQ) was proposed by William Stern as a normally distributed measure of intelligence with median 100 and standard deviation 15 (Stern & Whipple, 1914). Today, the IQ is a widely used and commonly computed directly from the results obtained on standardized psychological tests such as the Wechsler tests and Raven's Progressive Matrices (Groth-Marnat, 2009; Raven & Court, 2003). In using performance on such standardized tests to define intelligence, implicitly intelligence can be measured on any cognitive agent; either human or machine (Bringsjord & Schimanski, 2003).

The view that intelligence can be defined in terms of performance on standardized psychological tests has been challenged by several authors. Legg and Hutter (2007) argues that an ad hoc computer program that performs well on some specific intelligence tests would not qualify as intelligent: according to their paradigm, passing such test is a necessary, but not sufficient condition for intelligence.

[*] Corresponding author.
E-mail address: claes.strannegard@gu.se (C. Strannegård).

1389-0417/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.cogsys.2012.08.002

The term *artificial intelligence* (AI) was coined by John McCarthy, who defined it as "*the science and engineering of making intelligent machines*" (McCarthy, 2007). Whatever the exact relation between performance on IQ tests and human intelligence might be, constructing computer programs for solving standardized intelligence tests is a fundamental challenge to AI. As we shall shortly see, this challenge has been met only in part.

1.2. Raven's Progressive Matrices

Raven's Progressive Matrices (RPM) is a standardized intelligence test that was introduced by Raven (1936). Each RPM problem is presented as a 2×2 or 3×3 matrix of images following some pattern. The bottom right position, or cell, is left blank and the solver's task is to choose the missing image from a set of eight solution candidates. Fig. 1 shows an example of this type of problem. For copyright reasons, the illustrations in this paper do not depict original RPM problems, but constructed equivalents.

Extensive studies indicate high levels of correlation between the RPM scores and the scores on a range of other tests of (general) intelligence (Raven & Court, 2003, Snow, Kyllonen, & Marshalek, 1984). This, together with the fact that the RPM tests are entirely picture-based and require

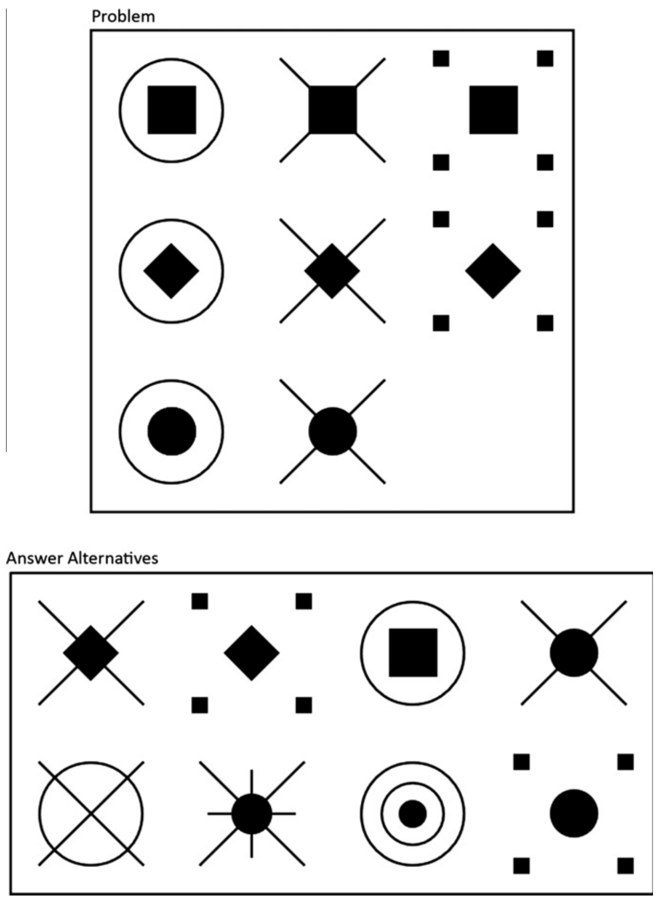

Fig. 1. Example of a 3×3 RPM-like problem.

no language proficiency, has helped build the popularity of the tests.

On an abstract level, RPM problems are similar to pattern recognition problems since they are both problems of inductive reasoning, alike to number progression problems (Varzi, 2006). Moreover, pattern recognition is closely related to perception, particularly to how objects and groups of objects are perceived. Such phenomena have been studied extensively in Gestalt theory (Wertheimer, 1939) and structural information theory (SIT) (Leeuwenberg, 1968, 1971).

The RPM set includes the Standard Progressive Matrices (SPM) and the Advanced Progressive Matrices (APM). The SPM, which is the target of this work, consists of five (increasingly difficult) sets of questions labeled A–E (Raven & Court, 2003). Each set contains 12 matrices. The sets A and B consist of 2×2 matrices, while C, D and E consist of 3×3 matrices. Another widely used set of RPM is the Colored Progressive Matrices (CPM), aimed at children, the elderly and people with learning disabilities.

1.3. Previous work

The computational aspects of RPM have been subject to considerable investigation. Carpenter, Just, and Shell (1990) created two RPM solvers aimed at modeling the RPM solving performance of average and high scoring human subjects tested earlier in their study. Both programs achieved results correlating well with experimental data, however the input to the solvers were textual descriptions of the RPM problems, meaning that none of them solves RPM problems strictly speaking (Meo, Roberts, & Marucci, 2007).

Bringsjord and Schimanski (2003) implemented an RPM solver based on an automatic theorem prover, but reported no results on any of the sets.

Lovett, Forbus, and Usher (2007, 2010) developed an RPM solver using the SME/CogSketch sketch understanding architecture (Forbus, Lockwood, Klenk, Tomai, & Usher, 2004; Forbus, Usher, Lovett, Lockwood, & Wetzel, 2008). They used an analogical reasoning strategy and addressed sections B-E of SPM with a score of 44/48 correct solutions.

Kunda, McGreggor, and Goel (2010) implemented two different visual solution strategies, of which the best performing one solves 58% of the SPM set.

Rasmussen and Eliasmith (2011) proposed a neural model for inductive rule generation and successfully applied it to RPM but did not report detailed results on the problem set.

1.4. Focus of this work

In order to produce an answer to each matrix, all of solvers mentioned above rely on, or at least make use of, the eight provided solution candidates. Such practice is perfectly legitimate, as access to the alternatives is part of the format of the RPM test. However, evidence indicates that high-achieving human solvers employ strategies that take the solution candidates into less account, or none at all except from finally comparing "their" solution to the answer alternatives, than those used by lower scoring subjects (Bethell-Fox, Lohman, & Snow, 1984; Carpenter et al., 1990; Vigneau, Caissie, & Bors, 2006). To what extent are then the answer alternatives *needed* in order to give a correct answer?

In this work we will present an architecture capable of producing and drawing RPM solutions without considering the answer alternatives.

2. Method

We will now detail our approach to solving RPM computationally, after specifying the problem at hand.

2.1. Patterns

In the instructions to the RPM test, the participants are asked to pick the answer that "completes the pattern". They are also informed that each problem has exactly one correct solution candidate. These instructions leave two fundamental questions unanswered:

1. What counts as a pattern?
2. If more than one pattern applies, which one should be preferred?

Patterns have been extensively studied in many research fields, including art and design theory, mathematics and computer science. For instance, different notions of pattern are considered in information theory, number theory, graph theory and image processing. As far as we understand, none of the standard notions of pattern in these research fields seem to fit the context of RPM problems. Therefore, the RPM instructions might seem unclear both to the specialist and the layman.

For instance, various notions of pattern that are based on Kolmogorov complexity (in its original form) do not seem to fit. The Kolmogorov complexity of a computer representable object is roughly the length of the shortest program that generates it (Li, 1997). In the context of RPM problems, this complexity measure has two serious drawbacks: (i) it depends on the language that the programs are written in and (ii) it is not computable (because of the halting problem).

Nevertheless we can certainly assume some intuitive understanding of patterns and consider the question of preference among those patterns. A classical criterion for preferring certain patterns or explanations over others is simplicity (Chater & Vitányi, 2003). This idea goes back to Occam, Kolmogorov and Solomonoff, among others (Li, 1997). The term simple is used with a large number of different meanings in several subdisciplines of mathematics and computer science. Again it is not clear to us which one of these notions of simplicity, if any, would fit in the context of RPM problems.

Chater (1999) considers simplicity to be a subjective notion that depends on the cognitive agent. This is in line with everyday usage of the term, but it is in contrast to the idea of simplicity as a Platonistic mathematical concept. Such subjective notions of simplicity are considered in simplicity theory (Chater, 1999; Chater & Vitányi, 2003; Dessalles, 2010).

2.2. Progressive matrices as a computational problem

Our conclusion from the above analysis is that, while the RPM test (pick the correct answer alternative) is unambiguous, the task of solving the RPM matrices alone (without alternatives) is not well-defined.

The two problems are undoubtedly related, but they are computationally distinct: Finding the best solution from a set of solutions is an *optimization problem* while problems where such set is not provided and whose output is more complex than a yes/no answer are called *function problems*.

Assuming an optimal solver able to extract and process all the available information, solving an optimization problem requires the input to provide sufficient information to *rule out all solution candidates but one*. In a function problem however, the input has to provide all the necessary information needed to *compute the solution*.

To give an example: consider a progressive matrix whose cells contain circles of radii 1, 2, 3 on row 1; 2, 3, 4 on row 2 and 3, 4 on row 3; suppose also that the centres of the circles are randomly placed. As there is no pattern governing the centre position, without solution candidates such problem does not admit a uniquely determined solution but an infinite number of them, i.e. the problem is not well-defined. However, assume the same matrix is presented with answer alternatives of which only one is a circle of radius 5. Now the problem admits a unique solution and the additional information required to determine it is provided within the alternatives.

Conversely, assume now a progressive matrix solvable and presented without alternatives: if a (non-optimal) solver is not able to extract and process enough information to produce a solution, it simply fails; if the matrix is instead presented with alternatives and the amount of information the solver is able to gain is enough to single out the right answer, it succeeds. Therefore it stands that:

- A given progressive matrix can or can not be solvable depending on whether is presented with or without solution candidates.
- Solving a progressive matrix without alternatives, if possible, is a harder problem (requiring more information to be extracted) than solving the same matrix with alternatives.

2.3. Strategy outline

Since our goal is to investigate the solvability of the RPM test items without utilizing the alternatives, we will provide explicit answers to the questions (1) and (2), stated in the *Patterns* section, as our starting points, thus giving a well-defined meaning to the RPM problems making the only factors in play:

- The repertoire of patterns known by the solver
- The amount of information provided by the progressive matrix

Our basic strategy for solving RPM problems is to mimic successful human problem solving strategies. Thus we exploit the fact that RPM problems are solvable by humans. More precisely, we are going to build a simple cognitive model for the purpose of problem solving, following the strategy of anthropomorphic reasoning (Strannegård, 2007).

The cognitive model was mainly obtained through introspection of our own problem solving strategies. It features an ontology of objects divided into five levels of abstraction. The abstraction levels were inspired by Gestalt theory and SIT. The ontology also features a small set of patterns that apply on different abstraction levels and a preference order on those patterns. As Rasmussen and Eliasmith (2011) showed, patterns can not only be part of the subject's previous knowledge but they are also possible to induce and generalize when solving the RPM test. Pattern origination (recall from memory vs on-spot induction) then is a separate cognitive task from the generation of RPM solutions and therefore the two can be independently modeled and addressed.

Our program takes as input progressive matrix problems represented in a generic vector graphics format. Thus we use a complete graphical representation of the problem, similar to Lovett et al. (2007), but without requiring any prior qualitative labeling or specific grouping of the graphical elements. The program does not take any representation of the answer alternatives as input, thus computing the solutions on the basis of the progressive matrices alone. The program generates solutions in the form of images encoded in the same vector format as the input and draws them on-screen, making it possible to unambiguously tell whether the generated solution is the correct one.

2.4. Knowledge representation

The first part of the designed cognitive model is the ontology of concepts providing a way to formalize progressive matrices.

2.4.1. Input format

Sections C–D–E of Standard Progressive Matrices were reproduced in the chosen vector graphics format; each one of the input files encodes one problem. In vector graphics, visual elements are encoded as vectors of attribute–value pairs.

2.4.2. Abstraction levels

Progressive matrices can be described in terms of the following abstraction levels:

- *Attributes* – Isolated properties of graphical elements, e.g. height, width, color, position.
- *Elements* – Single graphical elements, e.g. a filled black circle.
- *Groups* – Sets of distinct elements behaving as a single one, such as two lines making up an X or a + figure.
- *Cells* – All the graphical elements contained in a progressive matrix cell.

2.4.3. Representation structure

The concepts introduced above are implemented using a hierarchical graph structure: each layer corresponds to an organizational level of the input problem and each node corresponds to a feature on that level. The thought behind this type of approach is being able to detect patterns at the most abstract level at which they occur without discarding the information needed to go back to a graphical representation, allowing the computed solutions to be drawn back into an image once computed.

Additional node layers dedicated to positioning information were introduced. This separates the graphical elements from their placement within a cell, introducing a visual/spatial distinction, splitting the information load of the two components. Table 1 presents the node hierarchy of the representation graphs, in order of ascending abstraction.

Element Models and *Groups* are not present in every progressive matrix; therefore layers 3, and 5 and 6, are populated only when their presence is detected and are otherwise empty.

Fig. 2 shows a simple progressive matrix and the representation structure the program extrapolates from it; for visual clarity reasons, however, the diagram is simplified: it does not depict all the considered attributes and the *Position* attribute is presented as a single, qualitative one instead of two quantitative coordinates. The *CellObject* nodes of the diagram correspond to *Absolute Position* nodes of the actual structure.

In Fig. 3 we have the graph representation for a single cell of a more elaborate progressive matrix. The four squares at the corners constitute a *Group*; note how all of them are represented by the same *Element* node and the different positions are introduced separately, at successive layers for the elements and the group respectively. *Group Element* nodes in the diagram correspond to *Relative Position* nodes in the actual structure.

2.4.4. Structure computation

The graph structure is created only from the information contained in the input and it undergoes subsequent expansion in several stages. The algorithms involved in the process are deterministic, so the same input file will always yield the same representation structure. The chart in Fig. 4 shows an overview of the process:

1. *Initialization:*The vector graphics information within the input file is read and processed: for each graphical object the names and the values of some predefined attributes

Table 1
Graph node layers.

Number	Name	Description
1	Attributes	Relevant attributes directly extracted from the vector graphics objects (shape, line thickness, ...) or simple geometrical properties computed from them (bounding box width, center position, ...)
2	Attribute Values	Values of the extracted attributes, e.g. type = rectangle, line thickness = 2
3	Element Models	Sets of *Attribute Values* representing maximum common denominators with respect to the *Element Instances*
4	Element Instances	Positionless single visual elements, constituted by a combination of links to *Attribute Values* and *Element Models*
5	Relative Positions	*Element Instances* positioned relatively to the other ones belonging to the same group
6	Groups	Positionless aggregates of *Relative Positions* comparable and processable together with single *Element Instances*
7	Absolute Positions	*Element Instances* or *Groups* positioned inside the *Cell*
8	Cells	The actual cells of the progressive matrix, containing a number of *Cell Objects*, which are links to the relevant *Absolute Positions*

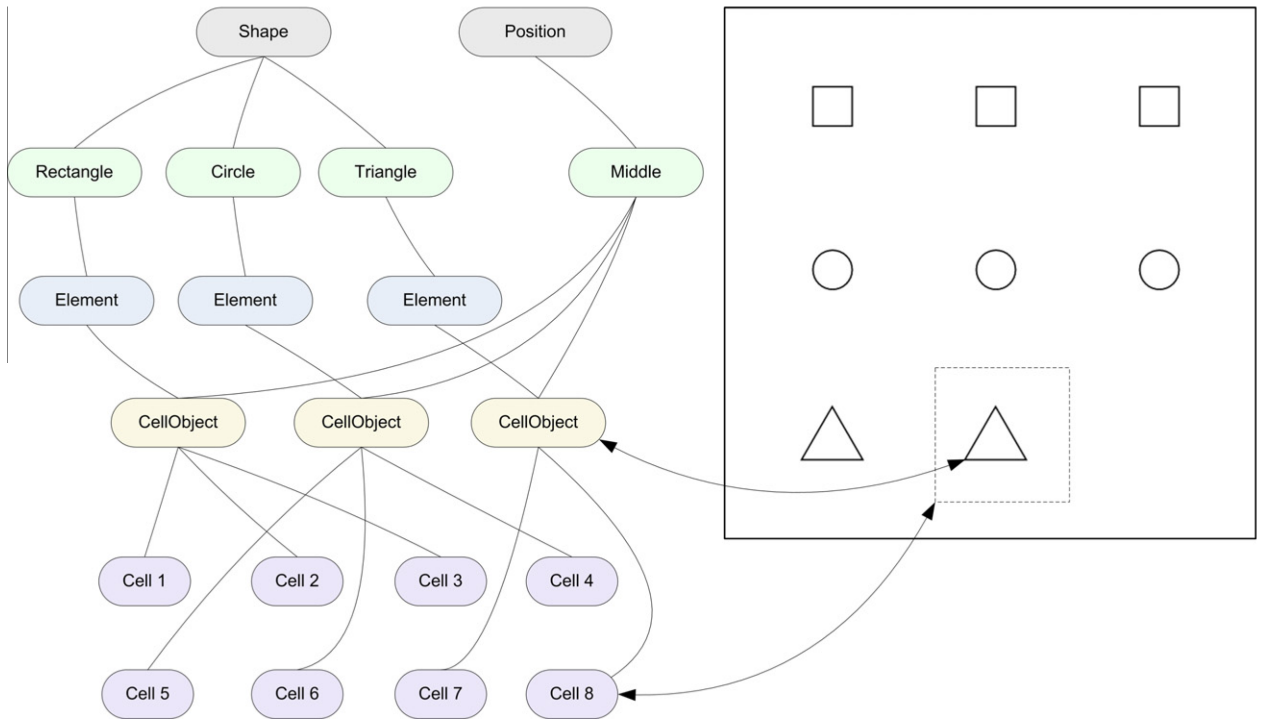

Fig. 2. Representation structure for a simple progressive matrix. Three types of elements can be seen – rectangle, circle and triangle. They are all positioned in the Middle, thus requiring only one attribute node for Positioning, creating one positioned CellObject each. Each CellObject is then placed into three cells each, except for the Triangle which is only present in two cells in the problem.

are extracted and used to populate and interconnect layers 1, 2, 4, 7, and 8. The chosen vector graphics format, *XAML*, features both specific (Ellipse, Rectangle, ...) and generic (Polygon, Polyline, ...) shape classes. In our input files we use the specific ones when appropriate and the generic ones otherwise, for clarity and readability. Regarding segmentation of “compound” visually connected shapes, such as the ones in the middle column of Fig. 3, the program is able to function with different variants of it (e.g. two lines and a third shape at the center vs a single closed polyline) as long as the same policy is applied throughout the input problem file. No preprocessing operations affecting visual segmentation are performed.

2. *Common attributes extraction:*Attributes having only one value throughout the entire problem are removed from the structure since they can be considered constants, providing no information about the solution pattern. Later, these common attribute values will be added to those of each generated solution element.
3. *Element instances cleanup:*Duplicate *Element Instance* and *Absolute Position* nodes are eliminated. From here on all the identical-looking graphical elements in the problem are represented by a single *Element Instance* in the structure; and for each one of them a single *Absolute Position* node represents all the instances located at the same position within their respective cell (s).

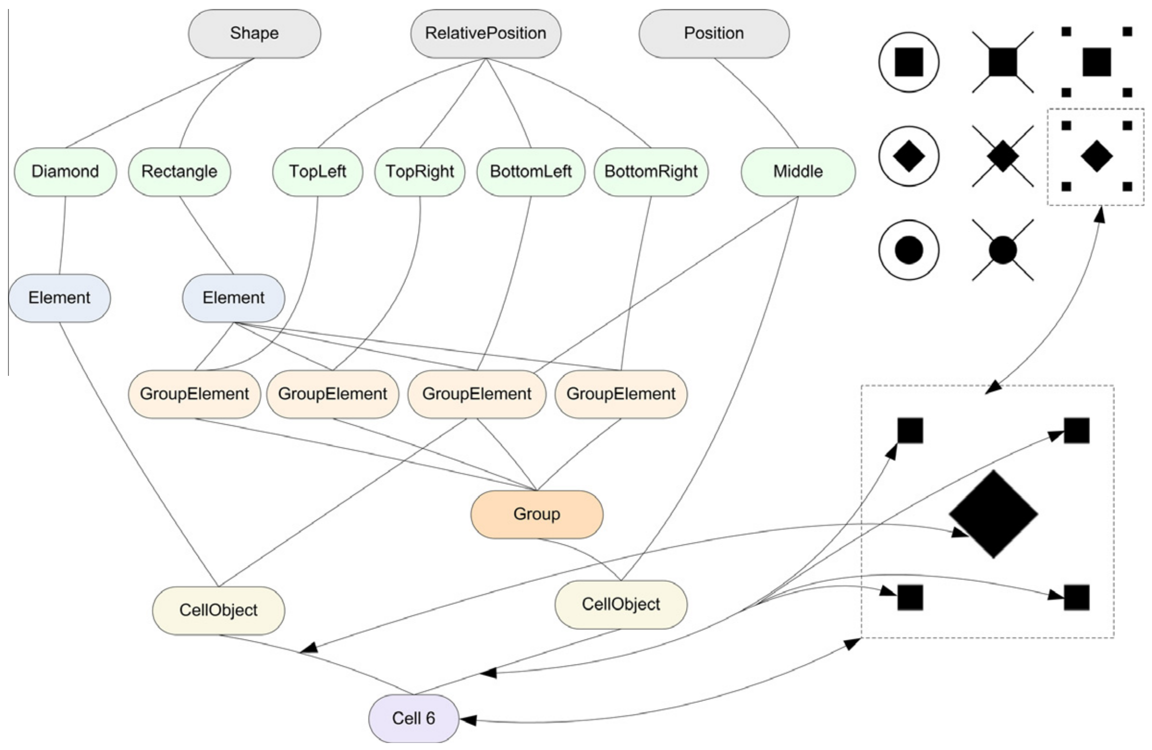

Fig. 3. Representation structure for a single cell of a moderately elaborate progressive matrix. Two element types can be seen, Diamond and Rectangle. The diamond is placed in the middle of the cell, similar to the previous figure. The rectangle occurs four times in different positions. In early stages of processing those four positioned rectangles were represented as four separate CellObjects (with different positions), but after the input is fully processed they are grouped together into one group, since they always occur together. The group is placed in the center of the cell, and the rectangles are represented as GroupElements (pointing at the same rectangle element) having a RelativePosition (relative to the center of the group).

4. *Element models creation:*Population of layer 3. *Attributes* considered for model creation do not include those related to positioning. The purpose of *Element Models* is to provide a way to safely infer additional *Attribute Values* given a single one.
5. *Common elements extraction:Absolute Position* nodes connected to all eight problem cells are removed from the structure and added to the objects of the solution cell.
6. *Element groups creation:*Population of layers 6 and 7. Groups are computed on the basis of *Absolute Positions* co-occurrence; no information about groups is provided in the input. Two or more *Absolute Position* nodes are grouped if and only if they appear together in the same cells throughout the problem. Grouped nodes are replaced by a *Relative Position* and a new *Absolute Position* referring to the centre of the whole group is generated.
7. *Group merging:*Since the employed grouping algorithm works only on the basis of co-occurrence; this can lead to situations where multiple compound visual objects are grouped together, but elsewhere a separate group for the single compound visual object has been constructed as well. These situations are resolved by replacing *Relative Positions* representing occurrences of minimal groups within larger, non-minimal, ones with *Absolute Positions* referring to the minimal group.

2.5. Computational model

The proposed solving strategy relies on pattern matching. Problems can be processed either row or column-wise; patterns holding for both of the first two are applied to the third one, obtaining a prediction value for the solution cell. The other principle behind the strategy is the use of different abstraction levels in order to solve the problems by processing them in as little detail as possible.

2.5.1. Abstraction levels

The levels employed in the pattern matching stage are related to, but slightly different, from those featured in the representation structures:

1. *Cells* – entire content of a cell
2. *Object Count* – number of objects within a cell
3. *Cell Objects* – objects positioned within a cell
4. *Instances* – positionless visual objects
5. *Element Models* – sets of correlated attributes

2.5.2. Pattern detection

The patterns are built as entities consisting of two distinct parts: the algorithm responsible for the matching process and the proper pattern function.

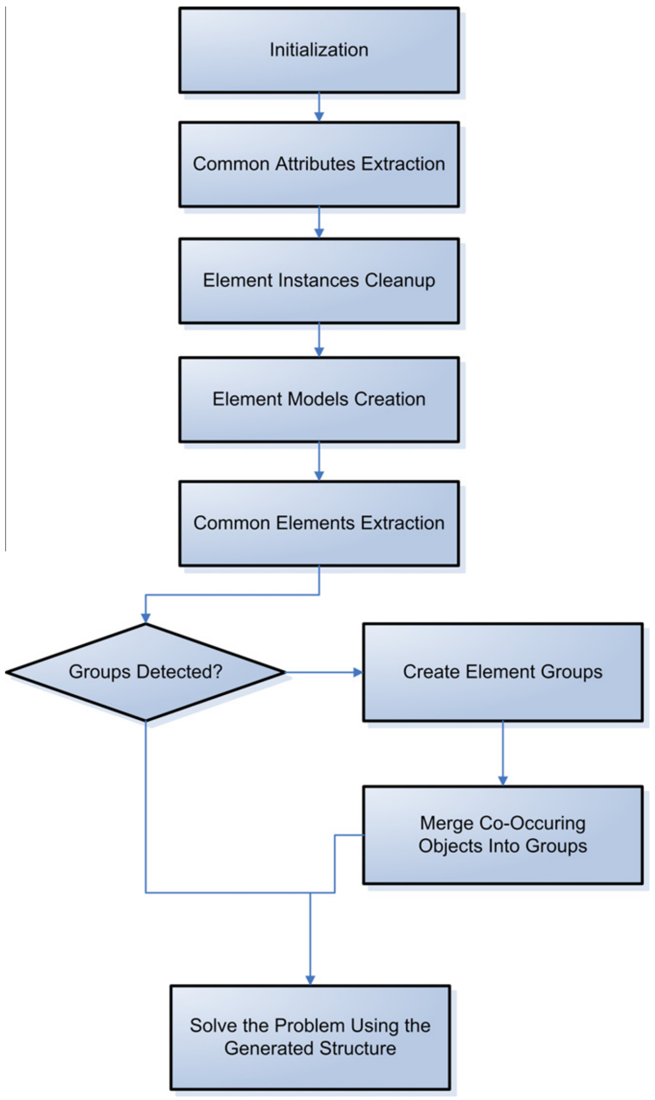
Initialization
Common Attributes Extraction
Element Instances Cleanup
Element Models Creation
Common Elements Extraction
Groups Detected?
Create Element Groups
Merge Co-Occuring Objects Into Groups
Solve the Problem Using the Generated Structure

Fig. 4. Graph structures generation steps.

The *Algorithm* is aware of which of the abstraction levels are relevant for the underlying pattern, it iterates the graph structure and it provides input to the *Pattern Function*.

The *Pattern Function* receives an ordered list of objects and it returns whether the pattern holds or not. If it holds and the parameters correspond to a third row or column, the function also returns a prediction value for the solution cell.

2.5.3. Available patterns

Currently there are seven implemented patterns; the architecture is very modular and additional ones can easily be added. Table 2 lists the implemented patterns while Table 3 shows the abstraction levels each pattern is capable of operating on. The patterns are similar to the "rules" used by Carpenter et al. (1990) and are consistent with those the system described by Rasmussen and Eliasmith (2011) is able to autonomously induce.

2.6. Solution process

When a pattern fits the problem it outputs a part of the solution for the missing cell which is added to the representation structure; the level of these partial solutions is the same as where the pattern was found. This means that if a pattern matches on the *Cell* level, the entire content of the solution cell has been predicted and the solution process can stop. If, on the other hand, a *Cell Object* is predicted, the solution might not be complete yet – in this case the process stops when the number of *Cell Objects* predicted on the *Objects Count* abstraction level has been generated.

The process of solving a progressive matrix is then defined as iterating through the patterns until the missing cell has been completely determined or until all the patterns have been tried on all the abstraction levels. The patterns are iterated according to the order they are listed in Table 2, while the abstraction levels are processed following the disposition presented in Page 12. The processing direction preference is horizontal first, vertical second.

2.7. Implementation

The described structures and algorithms were implemented as a Windows application written in C#.Net version 3.5, while the chosen vector graphics format for the input files was XAML. When an input file is loaded the program displays the progressive matrix and launches the computation of the solution. At the end of this operation, either because of successful completion or search space exhaustion, the computed solution is drawn in its intended position: the bottom-right cell of the matrix. The application also renders drawings of the graph representation structures.

3. Results

Given that the program produces outputs in form of on-screen images and it is built on the premise of not having access to the solution alternatives, the computed and displayed solutions are evaluated by visually comparing them to the answer alternatives of the original RPM test items.

It is also trivial to, after representing the correct solution candidate in the same format as each cell in the problem set, compare the given solution to the correct one, one vector graphics element at the time, to make sure that the given solution is indeed an exact representation of the correct solution.

Information about the occurred pattern matches is taken from the execution logs and messages produced by the application.

Table 5, featured at Page 21 details all the obtained results. Our program solves 8 of 12 SPM set C problems, 10 of 12 for set D, and 10 of 12 for set E. In total 28 of 36 problems are completely solved (78%). For four problems partial solutions were obtained and for the remaining

Table 2
Available patterns.

Name	Matches...
Identity	... When the passed-in entities are equal
One of each	... When three entities or entity groups occur once per every row or column, not necessarily in the same order
Numeric progression	... When for each set of passed-in integers, the differences between all pairs of integers are equal
Translation	... When the passed-in *Absolute Position* nodes point to the same instance node (*Element Instance* or *Group*) and the displacement of *x* and *y* coordinates from each cell to the next one is equal
Binary AND, OR, and XOR	... When two of the *Cells* can be seen as the operands and the remaining one as the result of the logical operation, in any order

Table 3
Abstraction levels each pattern can operate on.

Pattern	Abstraction level				
	Cells	Object count	Cell objects	Instances	Element models
Identity	✓	✓	✓	✓	✓
One of each	✓	✓	✓	✓	✓
Numeric progression		✓			
Translation				✓	
Binary AND	✓				
Binary OR	✓				
Binary XOR	✓				

four the program was not able to compute any solution element. The total number of pattern matches across the SPM set exceeds those of the problems because in several cases more than one pattern is used to produce a complete solution. In no case the program produced wrong pattern matches or solutions.

3.1. Performance

3.1.1. IQ scoring

Evaluation of the program's performance produced an IQ of 100 (Raven, Raven, & Court, 2004), assuming a perfect score on SPM sections A and B, which is a common situation for human subjects achieving the same results as the program in sections C, D, and E (Sara Henrysson Eidvall, Chief Psychologist of Mensa International, personal communication).

3.1.2. Computation Times[1]

Considering only the time spent for the solution search the average problem is solved in under 10 ms, the hardest takes around 55 ms, and the easiest one less than 4 ms. The total solution search time for the problem set is less than 500 ms.

3.2. Patterns

Most of the problems can be solved using a very small subset of the implemented patterns. The distribution of the patterns across the sets is worth noting as well. The pattern requirements of section C are quite diversified; however in section D 9 out of the 10 solved problems need the *One of Each* pattern, and in 11 out of 12 the *Objects Count* is predicted using *Identity*. For the E set, 8 out of the 10 solved problems only require *Binary XOR*.

3.3. Abstraction levels

In 22 out of 36 problems the number of the graphical objects of the solution cell (patterns on the *Objects Count* level) is predictable separately from any other kind of information about them. Examining the others abstraction levels, 14 of the 28 solved problems are determined on *Full Cell*; 13 are solved on the *Cell Objects* level, of which 3 feature two distinct patterns; only one problem is solved on the *Element Models* level.

3.4. Pattern preference relation

In addition to the processing order described in Table 2, the program was run using three other pattern orders:

- *Alternative 1*: *Binary XOR* is processed before the two other binary patterns.
- *Alternative 2*: *One of Each* applied before *Identity*.
- *Alternative 3*: Combination of Alternatives #1 and #2.

Table 4 presents the pattern match results for these sequence variations. Changing the order of pattern iteration did not alter the number, final appearance or accuracy of the produced graphical solutions but it affected their pattern composition. *One of Each*, when prioritized, took over most of the patterns usually solved via *Identity* because the two are often present together: one horizontally and the other vertically. Prioritizing *Binary XOR* solves all the *Bin-*

[1] On a Core i7 920 @ 2.67 GHz, 12 GB RAM, Windows 7 64 bit.

Table 4
Cumulative sums of pattern occurrences for the original and alternative function orders.

Order	Pattern						
	Identity	One of each	Numeric progression	Translation	Binary AND	Binary OR	Binary XOR
Original	17	13	4	3	1	4	5
Alternative 1	17	13	4	3	1	0	9
Alternative 2	3	20	4	3	1	4	5
Alternative 3	3	20	4	3	1	0	9

ary OR patterns because the two functions are equivalent for input pairs different than {1, 1}, of which there is no instance in SPM.

Worth noting is also that a very small number of patterns can solve a very large number of problems. Using our methodology and only the patterns *One of Each* and *Binary XOR* correctly solves as many as 21 problems of the total 36. Adding *Identity* solves another three problems, for a total of 24 correctly solved problems of 36 using *three patterns only*.

4. Discussion and conclusion

In this section we discuss why our program succeeds on certain problems and fails on others. We also discuss some aspects of the SPM test.

4.1. Solved problems

We defined an ontology containing objects, object attributes, object groups, patterns, and a preference relation on patterns. Based on this ontology, a program was developed that solves 28 of the 36 problems in the problem set.

Seven elementary patterns were implemented in our program: *Identity*, *One of Each*, *Numeric Progression*, *Translation*, *AND*, *OR*, and *XOR*. This small repertoire of patterns turned out to be enough for performing on an IQ 100 level on the SPM test.

The SPM set contains instances of patterns on all five levels of abstraction. Some of the SPM problems turned out to be solvable in more than one way. For instance, certain problems have a horizontal *One Of Each* pattern together with a vertical *Identity* pattern and can be solved using either one of them alone. Hence the pattern search order, or preference order, affects which ones are found (first). Another factor influencing the found patterns is the choice of processing directions. Since the algorithm is exhaustive, modification of these two settings results in different solution pattern composition without affecting the solver's overall performance.

4.2. Comparison with previous work

Compared to Carpenter et al. (1990), our solver identifies the same kind of abstract patterns as theirs, but starting from an unfiltered, drawable description of the problem.

With respect to Kunda et al.'s (2010) two visual solution strategies, our algorithm obtained better results on sections D–E and scored the same as their best performing algorithm on section C. However, our investigation had substantially different goals and ours is what they would classify as an "analytic" method: symbolic and with a structured knowledge model. In any case, both we and Kunda et al. reduce the solution of progressive matrices to a search on a well-defined space of possible solutions.

The work by Lovett et al. (2007, 2010) was tested on SPM sections B–E. They only report global results: 44/48 correct answers, however since they state their missed problems are the among the six hardest ones for humans, we assume none of them was part of section B; therefore obtaining a score of 32/36 on sections C–E, only two more correct answers than our solver.

Carpenter et al. (1990), Kunda et al. (2010), Rasmussen and Eliasmith (2011) start from a generated guess and pick the solution alternative minimizing a given comparison metric between the two; Lovett et al. (2010)'s model doesn't produce any prior guesses but instead assesses which one of several available strategies is to be applied to the RPM item, subsequently evaluates which of the eight candidates scores higher with respect to the chosen strategy and picks that as its answer.

Our solver is then the first program able to address the problem of solving progressive matrices without answer alternatives. Such problem, as shown earlier, is a different computational problem than the standard RPM test format, and a much harder one computationally.

4.3. Computational complexity

Our algorithm produces progressive matrices solutions in polynomial time, as the number of operations (comparisons) needed to verify a pattern holds throughout the matrix and output the corresponding element of the last cell is a polynomial function of the order of the matrix. Specifically it is of time complexity $O(n^2)$, where n is the matrix order.

Since the algorithm is deterministic, we can state that the function problem of finding solutions to well-defined progressive matrices is of complexity class FP by definition (Rich, 2008).

Our program fails to solve eight of the problems considered. The reasons for this fall into two categories.

4.3.1. *Unimplemented patterns*

Five of the unsolved problems could have been solved by our program if we had implemented some additional patterns that belong to the standard repertoire of image processing. These include morphing operations and boolean operations on the object level. Those patterns were not implemented due to time constraints.

Some other patterns, particularly *One Of Each* and *Identity*, keep recurring in the problem set with strikingly high frequency, cf. Table 5. Other patterns only appear once in the problem set. Those patterns we did not implement by choice.

4.3.2. *Underspecified patterns*

For three of the unsolved problems, the information available in the progressive matrix alone is not sufficient for producing an exact graphical rendition of the solution. Specifically, the missing information concerns the exact position, size or orientation of the graphical elements. These three progressive matrices then, are not well-defined problems if presented without answer alternatives and could never be solved fully by a solver *generating* the solution. Worth noting though is that although these problems can never fully be solved by our solution structure, our solver still creates partial solutions for some, predicting for example number of objects or shape types.

4.4. *The SPM test*

The conclusions about the SPM test that we draw from this study are as follows:

1. Strictly speaking, progressive matrix problems with or without answer alternatives, are not mathematically well-defined problems. We consider this to be a problem, since *a priori* it might be unclear to the test participants what kind of patterns they are supposed to look for.

Table 5
Detailed account of the obtained RPM solutions. For each problem the matches found by the program are indicated along with the abstraction level they were detected at, the levels are expressed using the numbered ordered described in Page 12.

Problem	Identity	One of each	Numeric progression	Translation	Binary AND	Binary OR	Binary XOR	Solved
C01	2	1						✓
C02								
C03						1		✓
C04	3							✓
C05	3		2					✓
C06	2							
C07	2			4				✓
C08			2					
C09	2			4				✓
C10				4				✓
C11			2					
C12		3	2					✓
D01	2	1						✓
D02	2	1						✓
D03		2, 1						✓
D04	2, 3	3						✓
D05	2, 3							✓
D06	2, 3	3						✓
D07	2, 3	3						✓
D08	2	5						✓
D09	2	3						✓
D10	2	3						✓
D11	2							
D12								
E01						1		✓
E02		2				1		✓
E03						1		✓
E04							1	✓
E05							1	✓
E06							1	✓
E07								
E08	2, 3							✓
E09					1			✓
E10		2					1	✓
E11							1	✓
E12								
Occurrences	17	13	4	3	1	4	5	28

2. Most of the matrices in the SPM set are solvable without any need for the provided alternatives, as demonstrated by our program. Since "response elimination" strategies are computationally less efficient (Bethell-Fox et al., 1984) this finding provides a very likely explanation for high-achieving solvers focusing less on the solution candidates, as observed in Carpenter et al. (1990) and Vigneau et al. (2006): high-achieving solvers are able to extract all information from the problem grid and construct a complete solution representation, this representation is then compared to the eight alternatives, looking for a perfect match.
3. It would be possible to construct a progressive matrices-based psychometric test without answer alternatives (provided the involved matrices are well-defined), and such test would be more challenging than RPM.
4. Introductory courses in computer science and mathematics at the university typically cover boolean operators, rotations, mirrorings, translations and methods of exhaustive enumeration and search. This is essentially all that is needed for performing well on the SPM test. This indicates that people with such an educational background have a considerable advantage on SPM and similar tests.
5. If it is true that RPM-like tests are biased with respect to educational background, then they could also be biased with respect to other sociological factors that correlate with educational background. Such factors could be e.g. gender, income, domicile, and cultural background.
6. It appears that training on RPM-like problems and not least reading research reports about them is an excellent preparation for the SPM-test.

4.5. Adaptability towards other RPM sets

The program and its underlying architecture are capable of working on the APM set, given they feature the same 3×3 structure as SPM C–E. With minor modifications it would also be possible to process the 2×2 SPM B. In order to produce correct results the appropriate pattern functions would have to be constructed, but the general representation structure and parsing algorithms would work as they are.

For the problems in SPM section A, the situation is different though: those are "gestalt" or "pattern completion" problems, not featuring a proper matrix structure with cells but being continuous visual patterns instead. Our program is not able to process such scenarios without heavy modifications to the perception stage.

Acknowledgment

The authors thank Professor Carl Martin Allwood at the Department of Psychology, University of Gothenburg and Sara Henrysson Eidvall at Mensa International for valuable input.

References

Bethell-Fox, C., Lohman, D., & Snow, R. (1984). Adaptive reasoning: Componential and eye movement analysis of geometric analogy performance* 1. *Intelligence, 8*(3), 205–238.

Bringsjord, S. & Schimanski, B. (2003). What is artificial intelligence? Psychometric AI as an answer. In *IJCAI'03: Proceedings of the 18th international joint conference on artificial intelligence* (pp. 887–893). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Carpenter, P., Just, M., & Shell, P. (1990). What one intelligence test measures: A theoretical account of the processing in the Raven progressive matrices test. *Psychological Review, 97*(3), 404–431.

Chater, N. (1999). The search for simplicity: A fundamental cognitive principle? *The Quarterly Journal of Experimental Psychology A, 52*(2), 273–302.

Chater, N., & Vitányi, P. (2003). Simplicity: A unifying principle in cognitive science? *Trends in Cognitive Sciences, 7*(1), 19–22.

Dessalles, J. -L. (2010, February). Simplicity theory – Unexpectedness. <http://www.simplicitytheory.org/>.

Forbus, K., Lockwood, K., Klenk, M., Tomai, E., & Usher, J. (2004). Open-domain sketch understanding: The nuSketch approach. In *AAAI fall symposium on making pen-based interaction intelligent and natural* (pp. 58–63).

Forbus, K., Usher, J., Lovett, A., Lockwood, K., & Wetzel, J. (2008). CogSketch: Open-domain sketch understanding for cognitive science research and for education. In *Proceedings of the fifth eurographics workshop on sketch-based interfaces and modeling*.

Groth-Marnat, G. (2009). *Handbook of psychological assessment*. New York: Wiley.

Kunda, M., McGreggor, K., & Goel, A. (2010). Taking a look (literally!) at the Ravens intelligence test: Two visual solution strategies. In *Proceedings of the 32nd annual conference of the cognitive science society* (pp. 1691–1696).

Leeuwenberg, E. (1968). *Structural information of visual patterns: An efficient coding system in perception*. The Hague: Mouton.

Leeuwenberg, E. (1971). A perceptual coding language for visual and auditory patterns. *The American Journal of Psychology, 84*(3), 307–349.

Legg, S., & Hutter, M. (2007). Tests of machine intelligence. In *50 Years of artificial intelligence: Essays dedicated to the 50th anniversary of artificial intelligence* (pp. 232–242). Berlin, Heidelberg: Springer-Verlag.

Li, P. V. M. (1997). *An introduction to Kolmogorov complexity and its applications*. Springer. <http://books.google.com/books?vid=ISBN0387948686>.

Lovett, A., Forbus, K., & Usher, J. (2007). Analogy with qualitative spatial representations can simulate solving Ravens Progressive Matrices. In *Proceedings from the 29th annual conference of the cognitive science society* (p. 449–454).

Lovett, A., Forbus, K., & Usher, J. (2010). A structure-mapping model of ravens progressive matrices. In *Proceedings of the 32nd annual conference of the cognitive science society* (pp. 2761–2766).

McCarthy, J. (2007, November). *What is artificial intelligence?* <http://www-formal.stanford.edu/jmc/whatisai/whatisai.html>.

Meo, M., Roberts, M. J., & Marucci, F. S. (2007). Element salience as a predictor of item difficulty for Raven's progressive matrices. *Intelligence, 35*(4), 359–368.

Rasmussen, D., & Eliasmith, C. (2011). A neural model of rule generation in inductive reasoning. *Topics in Cognitive Science, 3*(1), 140–153.

Raven, J., & Court, J. H. (2003). *Manual for raven's progressive matrices and vocabulary scales. Section 1: General overview*. San Antonio, TX: Harcourt Assessment.

Raven, J., Raven, J. C., & Court, J. H. (2004). *Manual for Raven's progressive matrices and vocabulary scales. Section 3. Standard progressive matrices 1992 edition including American and International Norms, 1992 adult norms*. Table SPM VI. Oxford: Oxford Psychologists Press Ltd.

Raven, J. C. (1936). *Mental tests used in genetic studies: The performances of related individuals in tests mainly educative and mainly reproductive*. Unpublished master's thesis, University of London.

Raven, J. C., & Court, J. H. (2003). *Manual for Raven's progressive matrices*. Research supplement no. 2 and Part 3, Section 7. San Antonio, TX: Harcourt Assessment.

Rich, E. (2008). *Automata, computability and complexity: Theory and applications*. Pearson Prentice Hall.

Snow, R., Kyllonen, P., & Marshalek, B. (1984). The topography of ability and learning correlation. In R. J. Steinberg (Ed.). *Advances in the psychology of human intelligence* (Vol. 2, pp. 47–103). Hillsdale, NJ: Erlbaum.

Stern, W., & Whipple, G. (1914). *The psychological methods of testing intelligence*. Warwick & York.

Strannegård, C. (2007). Antropomorphic artificial intelligence. In *Kapten mnemos kolumbarium* (Vol. 33). Göteborgs Universitetet. <http://www.phil.gu.se/posters/festskrift2/mnemo_strannegard.pdf>.

Varzi, A. C. (2006). Patterns, rules, and inferences. In J. E. Adler & L. J. Rips (Eds.), *Reasoning: Studies of human inference and its foundations* (pp. 282–290). Cambridge, UK: Cambridge University Press.

Vigneau, F., Caissie, A., & Bors, D. (2006). Eye-movement analysis demonstrates strategic influences on intelligence. *Intelligence, 34*(3), 261–272.

Wertheimer, M. (1939). Laws of organization in perceptual forms. In W. D. Ellis (Ed.), *A source book of gestalt psychology* (pp. 71–88). London: Routledge.