Project Work for the Course *Knowledge Representation and Reasoning, 2025* (Version 1)*

Jef Wijsen

Online on 10 March 2025

1 Introduction

Figures 1–3 show three pages of the book [Gol94]. Wikipedia has pages on the "Soma Cube" and other *solid dissection puzzles*.

Block puzzles, particularly those that involve finding a solution or arrangement of blocks, often fall into the class of NP problems:

- *Verification is easy:* For block puzzles, once you have a proposed solution (an arrangement of the blocks), it is usually easy to verify if it is correct. You can visually inspect the arrangement or apply simple rules to check if the blocks fit together as required.
- *Finding a solution is the challenge:* The challenge with block puzzles, similar to many NP problems, is that finding the solution (the correct arrangement) might be difficult and time-consuming. There might not be a straightforward algorithm that guarantees a quick solution, especially as the size or complexity of the puzzle increases.

Given that Answer Set Programming (ASP) is designed for modeling and solving problems in NP, it is well-suited for block puzzles. This alignment will be the focus in this project. Specifically, you are tasked with designing, analyzing, and solving your own block puzzle(s) using ASP.

The concluding paragraph in Fig. 3 reads:

> ***Golomb's challenge:*** "The reader is invited to look for additional simple solids into which the solid pentominoes will fit and, what is even more important, to discover other interesting shapes into which they can be assembled."

In this project, you are assigned the task of tackling Golomb's challenge. Specifically, you will create your own puzzle(s) that expand on Golomb's original challenge. Unlike Golomb's puzzle, you are not limited to using only solid pentominoes; you are encouraged to incorporate additional elements, such as colored blocks, as explained later in this report. However, it is important that Golomb's challenge remains a plausible special case of your puzzle, meaning your creation should still be conceptually connected to Golomb's challenge.

The remainder of this document is organized as follows. Sections 2 delves into distinct aspects of generating and solving block puzzles using ASP. This section aims to assist you in the creative process of designing your own block puzzle. Following this, in Section 3, you are asked to study the complexity of your puzzles. You should explore how the ASP programs for your block puzzle problem scale in practice, potentially supplemented with a theoretical analysis. Section 4 addresses the practical organization of this project, outlining the reports and oral presentation that have to be delivered. Section 5 lists the important dates.

2 Generating and Solving Puzzles with ASP

In this project, you are asked to design and solve your own block puzzles(s) in an automated fashion using ASP. This includes a number of steps, each with its associated task and possible outcome.

*New versions may be needed when students find mistakes in this project description or ask for additional clarifications.

SOLID POLYOMINOES

Numerous pentomino fans have suggested the possibility of gluing the twelve pentominoes together using five cubes instead of five squares each, and making solid patterns from them. More generally, one can list and enumerate the solid polyominoes of every number of sides. The solid polyominoes, allowing the pieces to be reflected and rotated, up through the tetrominoes, are shown in figure 112.

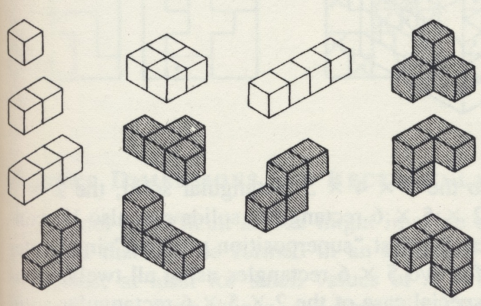


Figure 112. The lower-order solid polyominoes, with the Soma Cubes shaded.

Note that the first *nonplanar* figures (that is, those in which all the cubes do not lie in one plane) occur among the tetrominoes, since any three points lie in a plane, whereas four points (the centers of the four cubes) need not. Two of the three nonplanar tetrominoes are mirror twins, differing from each other as a right shoe differs from a left one.

A game invented by the contemporary Danish puzzle expert and game innovator Piet Hein and available commercially as the "Soma Cube" puzzle involves the seven solid polyominoes that are not simple rectangular solids (six tetrominoes and one tromino) shaded in figure 112. The object is to assemble these either into a $3 \times 3 \times 3$ cube, or into any of many other intriguing shapes. (A lengthy account of these problems is contained in the "Mathematical Games" column of *Scientific American* for September 1958.)

Figure 1: Page 79 of [Gol94]

If one uses the twelve planar pentominoes in their solid form, a natural problem is to fit them into a 3 × 4 × 5 rectangular solid. One of the many solutions to this problem is shown with its step by step construction in figure 113.

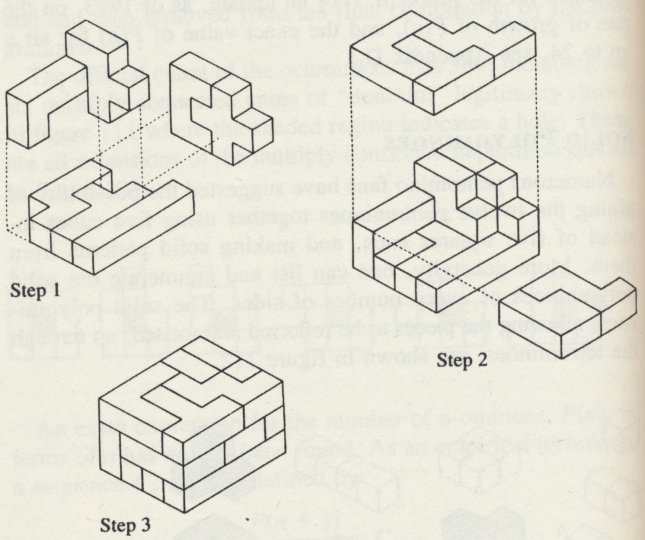


Figure 113. Construction of a 3 × 4 × 5 solid using the 12 solid pentominoes.

In addition to the 3 × 4 × 5 rectangular solid, the 2 × 3 × 10 and the 2 × 5 × 6 rectangular solids can also be constructed. In fact, the first "superposition problem" in chapter 2, which asks for two 5 × 6 rectangles using all twelve pentominoes, is a special case of the 2 × 5 × 6 rectangular-solid problem.

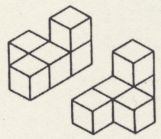
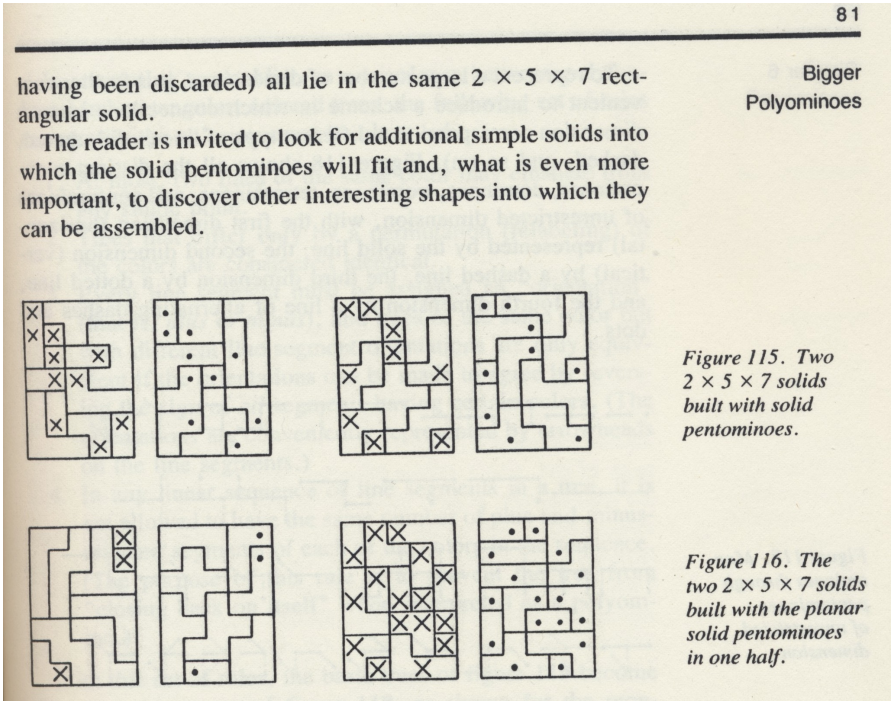


Figure 114. Mirror-twin solid pentominoes.

David A. Klarner has gone even farther, producing all twenty-nine solid pentominoes (or "pentacubes," as Klarner calls them) that are distinct under physical 3-dimensional rotations. (Thus, such mirror twins as those shown in figure 114 count as two separate pieces, since neither can be rotated into the other within the confines of 3-dimensional space.) Unfortunately, since 29 is a prime, the number of simple solids using all of the solid pentominoes is rather limited. However, Klarner found that if the 1 × 1 × 5 piece is omitted, the remaining twenty-eight pentacubes can be fitted together into two separate 2 × 5 × 7 rectangular solids. Klarner's solution is shown in figure 115 (a dot or cross represents whether a cube extends upward or downward, respectively). Another solution, discovered by the author, is shown in figure 116. It embodies the additional constraint that the eleven planar pieces (the 1 × 1 × 5 pentomino

Figure 2: Page 80 of [Gol94]



81

Bigger Polyominoes

having been discarded) all lie in the same 2 × 5 × 7 rectangular solid.

The reader is invited to look for additional simple solids into which the solid pentominoes will fit and, what is even more important, to discover other interesting shapes into which they can be assembled.

Figure 115. Two 2 × 5 × 7 solids built with solid pentominoes.

Figure 116. The two 2 × 5 × 7 solids built with the planar solid pentominoes in one half.

Figure 3: Page 81 of [Gol94]

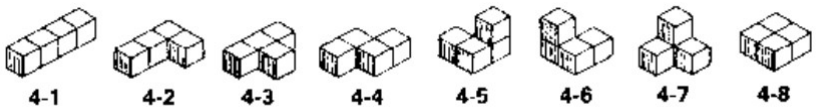


Figure 4: All the Possible Combinations of 4 Cubes.

2.1 Aspects of Puzzle Generation

Define Puzzle Elements: A block puzzle comprises a collection of building blocks, which may vary in shapes, colors, and properties.

Task: Define the elements of a block puzzle, including blocks, shapes, colors, and properties.

Outcome: Document detailing the defined elements with examples.

Establish Rules: Create rules for how these puzzle elements interact and fit together. Such rules can dictate how blocks can be rotated, stacked, or placed on a three-dimensional grid.

Task: Create rules for how your puzzle elements interact and fit together.

Outcome: List of rules with explanations and examples.

Constraints: ASP enables the specification of constraints to ensure generated puzzles are valid and interesting. Constraints can enforce rules like "no two identical shapes can be adjacent."

Task: Specify constraints using ASP to ensure generated puzzles are valid and interesting.

Outcome: Documented constraints with ASP code snippets.

Variations: A puzzle can have different variations and challenges, such as the creation of specific color patterns. For example, the building blocks of Fig. 4 could be colored in 6 different colors, where a block can have different colors on different faces. The objective may be to arrange these blocks in a cube with the additional difficulty that a same color cannot occur on two distinct faces of the cube. In another scenario, each building block could possess its own unique color, but blocks should be arranged in a cube such that adjacent blocks have distinct colors.

Task: Design variations your block puzzle, such as color patterns or unique block arrangements.

Outcome: Proposal for puzzle variations with ASP implementation examples.

Parameterization: With ASP, it is possible to develop puzzle generators that are parameterized, enabling the creation of puzzles of different sizes and complexities.

Task: Develop a parameterized puzzle generator using ASP.

Outcome: ASP program allowing generation of puzzles of varying sizes and complexities.

Randomization: Clingo's handling of randomness allows for the creation of randomized puzzles. This feature is illustrated by Example 2.1 for sudoku puzzles.

Task: Create a randomized puzzle generator.

Outcome: An effective procedure for generating randomized puzzles.

Example 2.1. The following ASP program aims at generating a sudoku puzzle by starting with a filled sudoku grid, using predicate `aa/3`, and then select from it 25 cells, using predicate `a/3`.

```
% This is SudokuGenerator.lp.
% Every cell stores at least one number.
1 { aa(I,J,K) : K=1..9 } :- I=1..9, J=1..9.
% Within each 3x3-square, every number occurs at least once.
1 { aa(I,J,K) : J=1..9 } :- I=1..9, K=1..9.
% Indices belonging to a Same Column.
isc(1,4). isc(4,7). isc(1,7).
isc(2,5). isc(5,8). isc(2,8). isc(3,6). isc(6,9). isc(3,9).
isc(I,J) :- isc(J,I).
isc(I,I) :- I=1..9.
% Indices belonging to a Same Row.
isr(1,2). isr(2,3). isr(1,3).
isr(4,5). isr(5,6). isr(4,6). isr(7,8). isr(8,9). isr(7,9).
isr(I,J) :- isr(J,I).
isr(I,I) :- I=1..9.
% Distinct cells in same row/column cannot store the same number.
eq(I,J,I,J) :- I=1..9, J=1..9.
:- aa(I,J,K), aa(II,JJ,K), isc(I,II), isc(J,JJ), not eq(I,J,II,JJ).
:- aa(I,J,K), aa(II,JJ,K), isr(I,II), isr(J,JJ), not eq(I,J,II,JJ).
% Select 25 cells.
25 {a(I,J,K) : aa(I,J,K)} 25.
```

```
#show a/3.
```

The above construction ensures that the ASP program on the slides available at https://web.umons.ac.be/app/uploads/sites/84/2024/06/ASP.pdf (slide 25) will find a stable model that fills the remaining $81 - 25 = 56$ cells. That ASP program can also be used to determine the number of stable models, which number should be 1 for a valid sudoku puzzle. Note also that clingo can be asked to generate different puzzles at random:

```
clingo -n 1 --rand-freq=1 --seed=5 sudokuGenerator.lp
```

By changing the value of `seed`, different puzzles will be generated. □

2.2 Aspects of Puzzle Solving

Solvable Puzzles and/or Unique Solutions: Clingo provides the capability to count the solutions for each puzzle generated. This allows designers to control the puzzle generation process, ensuring that each puzzle is solvable, while puzzles with an undesirably high number of solutions are avoided. Alternatively, designers can aim for puzzles with a single, unique solution.

Task: Create an effective procedure (such as an ASP program or Python script) to verify the solvability of generated puzzles. Additionally, if desired, ensure the uniqueness of solutions.

Outcome: Effective procedure for ensuring each puzzle has at least one valid solution, which may be required to be unique.

Optimal Solutions: ASP can be used to find optimal solutions to block puzzles. Designers can define optimization criteria (e.g., minimize moves, maximize score) within the ASP program.

Task: Define optimization criteria (e.g., minimize moves, maximize score) for a block puzzle.

Outcome: ASP program that finds optimal solutions based on defined criteria.

Analysis: Designers can assess the difficulty of a block puzzle by analyzing the solution steps or the number of solutions, or by measuring the efficiency of the clingo solver.

Task: Create an effective procedure (such as an ASP program or Python script) to analyze the difficulty of your block puzzle(s) based on parameters such as the sizes of the generated grounded programs or their execution times. Ideally, you should be capable of presenting puzzles with varying levels of difficulty, such as (1) Very Easy, (2) Easy, (3) Medium, (4) Hard, and (5) Harder.

Outcome: A method to assess puzzle difficulty and solver efficiency.

3 Theoretical Research Questions

Upon finishing the tasks detailed in Section 2, you are requested to conduct a scientific analysis of the complexity of the ASP programs that solve your puzzle problem. This requires that your puzzle problem be formally defined, followed by an experimental and theoretical analysis.

Problem Statement You should present a formal definition of the decision problem associated with your puzzle, which may have multiple variations. Appendix A serves as an example of how such a problem statement appears: it outlines the input (referred to as the "instance") and poses a yes-no question.

Running Experiments You are asked to answer the following question:

> How does your ASP solver scale in function of puzzle size? How do measures like the execution time and the size of the grounded program change as the puzzle size increases? Is the growth of the execution time polynomial (linear, quadratic, cubic, ...) or exponential?

The results from these experiments should be presented in both tabular and graphical formats; see Fig. 5 for an example graph.

Theoretical analysis Students who passed the course *Calculabilité et Complexité* are asked to study the complexity of the decision problem associated with their puzzle. Argue that your problem is in NP. To show that your problem is NP-complete, you may establish a reduction from one of the problems in Appendix A.

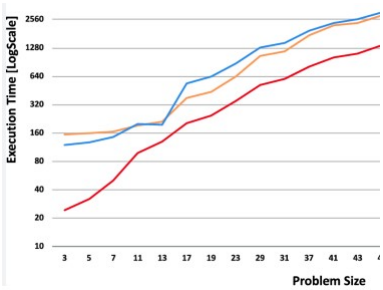


Figure 5: Execution Time as a Function of Problem Size.

4 Practical Organization and Deliverables

4.1 Groups

The project is conducted in **groups of two students**. Every group assigns a coordinator, who is responsible for all communications with the teacher (i.e., with me). At the beginning of the project, the composition of the groups is communicated by the coordinators to `jef.wijsen@umons.ac.be`.

4.2 Preliminary Report and Final Report

Two reports will have to be submitted:

1. A **preliminary report**, which should not exceed 5 pages, is expected to outline the core ideas and concepts of your project work. This submission, due by April 7, 2025, serves as an opportunity for each group to present their proposed ideas, methodologies, and initial plans for their project. The purpose of this document is to provide a concise yet comprehensive overview of the project's direction, goals, and intended approach. Feedback on this report will be provided before April 19, 2025, allowing teams to refine their ideas and strategies based on the input received.

2. A complete version of your **final report** of at most 12 pages is to be submitted in Moodle by May 19, 2025. In the week after your submission, I will read your report and give you some feedback. You can then submit a new version taking into account my remarks on the first version.

   Your report should begin with your findings concerning Golomb's challenge. After that, it should address the aspects (tasks and outcomes) discussed in Section 2. However, you are free (and even encouraged) to arrange and combine these aspects in whichever way you find suitable. You should also show how your ASP programs scale, as specified in Section 3. It is important that your report "tells a story" and, moreover, tries to "sell your story" to the reader. You should describe your block puzzles(s), argue that they are challenging and interesting, describe and discuss your solutions, illustrate solutions by examples, provide conclusions... I will assess not only the quality of your ASP programs but also the clarity and readability of your presentation.

4.3 Format

Both reports **MUST** (strong obligation) be formatted according the LIPIcs style and author instructions available at `https://submission.dagstuhl.de/series/details/LIPIcs#author`.

4.4 A Good Example

You may have a look at the textbook "Modelling Puzzles in First Order Logic" [Gro21] to find out how puzzles, and their solutions, can be presented in a user-friendly way. When you are within the domain `umons.ac.be`, you should have free access to this book at the following link. Otherwise get your copy on the Moodle pages of this course.

Imagine that together we are writing a new textbook "Modelling Puzzles in Second Order Logic," and that your puzzle constitutes a chapter of this book. Your chapter should be well structured and reader-friendly, with nice examples, as in [Gro21].

4.5 Oral Presentation

Each group will have a 20-minute slot to present their project, scheduled for Wednesday, May 7, 2025, and Thursday, May 8, 2025. The presentation must consist of precisely 10 slides (excluding the title slide), with each member responsible for presenting 5 slides, which do not need to be consecutive. Slides should be uploaded in Moodle before Monday, May 5 at 23H59.

4.6 Physical Realization (Optional)

You might want to construct an actual, physical version of your puzzle.

5 Important Dates

March 5, 2025: Start of project.

April 7, 2025, 23H59: Deadline for submission of Preliminary Report.

Before April 19, 2025: Feedback on Preliminary Report.

May 5, 2025, 23H59: Deadline for submission of slides for oral presentation.

Wednesday, May 7, and Thursday, May 8, 2025: Oral presentation.

May 19, 2025, 23H59: Deadline for submission of (first version of) Final Report.

June, 2025: Possibility to submit revised version of Final Report.

Importantly: do not hesitate to contact me if during your project work, you are blocked on some issue—I may be able to help you out.

References

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gol94] Solomon W. Golomb. *Polyominoes: Puzzles, Patterns, Problems, and Packings*. Princeton University Press, Princeton, New Jersey, 2nd edition, 1994.

[Gro21] Adrian Groza. *Modelling Puzzles in First Order Logic*. Springer, 2021.

A Some NP-Complete Puzzles [GJ79]

[GP11] **N×N GO (*)**

INSTANCE: Positive integer N, a partition of the "points" on an $N \times N$ Go board into those that are empty, those that are occupied by White stones and those that are occupied by Black stones, and the name (Black or White) of the player whose turn it is.

QUESTION: Does White have a forced win from the given position in a game of Go played according to the standard rules, modified only to take into account the expanded board?

Reference: [Lichtenstein and Sipser, 1978]. Transformation from PLANAR GEOGRAPHY.

Comment: PSPACE-hard.

[GP12] **LEFT-RIGHT HACKENBUSH FOR REDWOOD FURNITURE**

INSTANCE: A piece of "redwood furniture," i.e., a connected graph $G = (V,E)$ with a specified "ground" vertex $v \in V$ and a partition of the edges into sets L and R, where L is the set of all edges containing v (the set of "feet"), $R = E - L$, and each "foot" in L shares a vertex with at most one edge in R, which is its corresponding "leg" (not all edges in R need to be legs however), and a positive integer K.

QUESTION: Is the "value" of the Left-Right Hackenbush game played on G less than or equal to 2^{-K} (see [Conway, 1976] for the definition of the game, there called Hackenbush Restrained, and for the definition of "value")?

Reference: [Berlekamp, 1976]. Transformation from SET COVERING.

Comment: Remains NP-complete even for "bipartite" redwood furniture, but can be solved in polynomial time for the subclass of redwood furniture known as "redwood trees." As a consequence of this result, the problem of determining if player 1 has a win in an arbitrary game of Left-Right Hackenbush is NP-hard.

[GP13] **SQUARE-TILING**

INSTANCE: Set C of "colors," collection $T \subseteq C^4$ of "tiles" (where $<a,b,c,d>$ denotes a tile whose top, right, bottom, and left sides are colored a,b,c, and d, respectively), and a positive integer $N \leq |C|$.

QUESTION: Is there a tiling of an $N \times N$ square using the tiles in T, i.e., an assignment of a tile $A(i,j) \in T$ to each ordered pair i,j, $1 \leq i \leq N$, $1 \leq j \leq N$, such that (1) if $f(i,j) = <a,b,c,d>$ and $f(i+1,j) = <a',b',c',d'>$, then $a = c'$, and (2) if $f(i,j) = <a,b,c,d>$ and $f(i,j+1) = <a',b',c',d'>$, then $b = d'$?

Reference: [Garey, Johnson, and Papadimitriou, 1977]. Transformation from DIRECTED HAMILTONIAN PATH.

Comment: Variant in which we ask if T can be used to tile the entire plane $(Z \times Z)$ "periodically" with period less than N is also NP-complete. In general, the problem of whether a set of tiles can be used to tile the plane is undecidable [Berger, 1966], as is the problem of whether a set of tiles can be used to tile the plane periodically.

[GP14] CROSSWORD PUZZLE CONSTRUCTION

INSTANCE: A finite set $W \subseteq \Sigma^*$ of words and an $n \times n$ matrix A of 0's and 1's.
QUESTION: Can an $n \times n$ crossword puzzle be built up from the words in W and blank squares corresponding to the 0's of A, i.e., if E is the set of pairs (i,j) such that $A_{ij}=0$, is there an assignment $f: E \rightarrow \Sigma$ such that the letters assigned to any maximal horizontal or vertical contiguous sequence of members of E form, in order, a word of W?

Reference: [Lewis and Papadimitriou, 1978]. Transformation from X3C.
Comment: Remains NP-complete even if all entries in A are 0.

[GP15] GENERALIZED INSTANT INSANITY

INSTANCE: Finite set C of "colors" and a set Q of cubes, with $|Q|=|C|$ and with each side of each cube in Q having some assigned color from C.
QUESTION: Can the cubes in Q be stacked in one vertical column such that each of the colors in C appears exactly once on each of the four sides of the column?

Reference: [Robertson and Munro, 1978]. Transformation from EXACT COVER.
Comment: The associated two-person game, in which players alternate placing a new cube on the stack, with player 1 trying to construct a stack as specified above and player 2 trying to prevent this, is PSPACE-complete with respect to whether the first player has a forced win. INSTANT INSANITY is a trade name of Parker Brothers, Inc.