ASPECT: Answer Set rePresentation as vEctor graphiCs in laTex

Alessandro Bertagnon[1], Marco Gavanelli[2] and Fabio Zanotti[3]

[1]Dipartimento in Scienze dell'Ambiente e della Prevenzione, University of Ferrara, C.so Ercole I D'Este, 32, Ferrara, Italy
[2]Dipartimento di Ingegneria, University of Ferrara, Via Saragat 1, Ferrara, Italy
[3]Dipartimento di Informatica - Scienza e Ingegneria, University of Bologna, Viale del Risorgimento 2, Bologna, Italy

Abstract
Logic programming is a declarative programming paradigm that finds extensive use in the field of Artificial Intelligence (AI). As a result, it has become a valuable tool used in university courses for teaching students AI techniques. Besides Prolog language, the more recent Answer Set Programming (ASP) language turns out to be a powerful tool for developing advanced applications due to the expressiveness of the language and the availability of efficient solving systems. Unfortunately, the output of ASP solvers can be difficult to interpret, since it is a set of atoms, often long and verbose. This is most true in the case of students learning the language or in the case of experts developing applications for complex real-world problems. For these reasons, the ability to produce, when possible, a graphical representation of the solver output becomes useful to ensure easier interpretation of the results. In this paper we present ASPECT, a sub-language of ASP in which the user can directly define, in an intuitive and declarative way, a graphical representation of the answer set. The ASPECT atoms can be converted into the popular LaTeX markup language to produce vector graphics. The documents produced by ASPECT are easy to embed in documents such as scientific articles, course handouts, and presentations. Also, the development of user-friendly interfaces is critical for wider use of similar technologies in the industrial sector as well.

Keywords
Answer Set Programming, Answer Set Visualization, Vector Graphics, LaTeX

1. Introduction

Logic programming is a declarative programming paradigm that finds extensive use in the field of Artificial Intelligence (AI). As a result, it has become a valuable tool used in university courses for teaching students AI techniques [1].

Answer Set Programming (ASP) is a logic programming language with a semantics known as the stable models semantics [2]. The popularity of this language is due to its expressiveness and the availability of efficient solving systems [3, 4], which allow its use even in advanced applications. ASP programs are logic programs composed of facts and rules that represent the problem to be solved. The key idea of ASP is to model problems in such a way that its stable model(s) provides the solution(s). Stable models, also called answer sets, are represented by consistent sets of ground facts provided to the user through a textual representation often long

CILC'23: 38th Italian Conference on Computational Logic, June 21–23, 2023, Udine, Italy
alessandro.bertagnon@unife.it (A. Bertagnon); marco.gavanelli@unife.it (M. Gavanelli); fabio.zanotti2@studio.unibo.it (F. Zanotti)
0000-0003-2390-0629 (A. Bertagnon); 0000-0001-7433-5899 (M. Gavanelli); 0009-0004-7533-9751 (F. Zanotti)

and verbose. Unfortunately, text-based representation makes it difficult to interpret the answer sets, because important information is not easily identifiable. This problem equally impacts both students taking their first steps with ASP and experienced programmers when dealing with very complex applications.

In this paper we introduce ASPECT, a sub-language of ASP in which the user can directly define, in an intuitive and declarative way, a graphical representation of answer sets. This language complements other tools already presented for the graphical representation of answer sets [5, 6, 7, 8] but at the same time introduces some noteworthy advantages. First, ASPECT is designed to generate high-quality vector graphics. In fact, ASPECT atoms are converted to the popular markup language LaTeX making it easy to incorporate those graphics into documents such as scientific papers, course handouts and presentations. ASPECT also gives the programmer complete flexibility in graphics design since it provides only low-level graphical primitives (e.g., lines, polygons, ellipses, etc.) that can be exploited for any type of visualization. Finally, the syntax of ASPECT, although compatible with ASP, is strongly inspired by that of the popular TikZ language making its use more intuitive to those already familiar with the latter.

A preliminary version of the interpreter for the ASPECT language is available online at https://github.com/abertagnon/aspect.

The rest of the paper is organized as follows. First the ASPECT language and the current structure of its interpreter are described in Section 2. Some examples of possible uses of the ASPECT language are presented in Section 3. Related works are described in Section 4. The final section contains conclusions and some possible future developments of this project.

2. ASPECT

ASPECT is declarative sub-language of ASP that can be used to define the graphical representation of an answer set. ASPECT syntax consists of special atomic formulas that define rendering of geometric primitives such as points, lines, polygons, ellipses, etc. Currently, the language consists of 21 atoms that allow the user to represent 8 different geometric shapes and also manage their style properties such as color and fill (if applicable). The positioning of each element is determined by Cartesian coordinates. A complete list of ASPECT atoms is given in Tables 1 and 2.

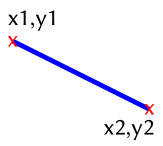
The syntax of the ASPECT language is inspired by the popular PGF/TikZ language developed for drawing vector graphics in the markup language LaTeX. Portable Graphics Format (PGF) is a language that provides a collection of low-level graphics primitive for LaTeX and TikZ is a set of higher-level macros that make PGF easier to use. To date, users who want to make graphics in LaTeX interact almost exclusively with TikZ, which has become a whole language of its own. ASPECT atoms can indeed be easily converted to the TikZ language. The ASPECT interpreter is responsible for the conversion.

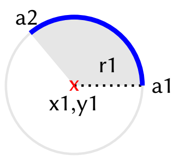
The ASPECT interpreter is written in Java, and it depends on an ASP solver and a LaTeX distribution. In the current implementation we chose `clingo`[1] as ASP solver and TeX Live[2] as LaTeX software distribution, but other systems can be supported with minor implementation

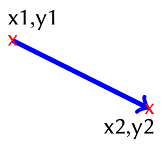
[1]https://potassco.org/clingo/
[2]https://www.tug.org/texlive/

Graphic element	ASPECT Atom	Parameters

node	`aspect_drawnode(x1,y1,"s1")`
	`aspect_drawnode(x1,y1,X)`
	`aspect_colornode(x1,y1,"s1",color)`
	`aspect_imagenode(x1,y1,image,width)`
		x1: x-coordinate
		y1: y-coordinate
		s1: text used to visualize the node
		X: ASP variable
		color: sets the line color
		image: path of image to include
		width: image width (px)

line	`aspect_drawline(x1,y1,x2,y2)`
	`aspect_colorline(x1,y1,x2,y2,color)`
	
		x1: start point x-coordinate
		y1: start point y-coordinate
		x2: end point x-coordinate
		y2: end point y-coordinate
		color: sets the line color

arc	`aspect_drawarc(x1,y1,a1,a2,r1)`
	`aspect_colorarc(x1,y1,a1,a2,r1,color)`
	
		x1: x-coordinate of the center
		y1: y-coordinate of the center
		a1: start angle
		a2: end angle
		r1: radius
		color: sets the line color

straight arrow	`aspect_drawarrow(x1,y1,x2,y2)`
	`aspect_drawarrow(x1,y1,x2,y2,color)`
	
		x1: tail x-coordinate
		y1: tail y-coordinate
		x2: tip x-coordinate
		y2: tip y-coordinate
		color: set the line color

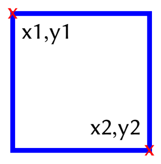
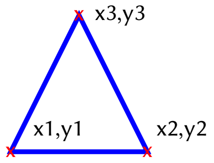
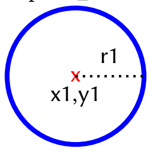
square\ rectangle	`aspect_drawrectangle(x1,y1,x2,y2)`
	`aspect_colorrectangle(x1,y1,x2,y2,color)`
	`aspect_fillrectangle(x1,y1,x2,y2,fill)`
	
		x1: first corner x-coordinate
		y1: first corner y-coordinate
		x2: second (opposite) corner x-coordinate
		y2: second (opposite) corner y-coordinate
		color: sets the line color
		fill: sets the fill color

Table 1
ASPECT syntax (part 1). The numerical values (coordinates) must be within the range 000-999 as required also by the TikZ language.

Graphic element	ASPECT Atom	Parameters

triangle	`aspect_drawtriangle(x1,y1,x2,y2,x3,y3)`
	`aspect_colortriangle(x1,y1,x2,y2,x3,y3,color)`
	`aspect_filltriangle(x1,y1,x2,y2,x3,y3,fill)`
	
		x1: first vertex x-coordinate
		y1: first vertex y-coordinate
		x2: second vertex x-coordinate
		y2: second vertex y-coordinate
		x3: third vertex x-coordinate
		y3: third vertex y-coordinate
		color: sets the line color
		fill: sets the fill color

circle	`aspect_drawcircle(x1,y1,r1)`
	`aspect_colorcircle(x1,y1,r1,color)`
	`aspect_fillcircle(x1,y1,r1,fill)`
	
		x1: center x-coordinate
		y1: center y-coordinate
		r1: radius
		color: sets the line color
		fill: sets the fill color

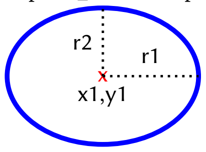
ellipse	`aspect_drawellipse(x1,y1,r1,r2)`
	`aspect_colorellipse(x1,y1,r1,r2,color)`
	`aspect_fillellipse(x1,y1,r1,r2,fill)`
	
		x1: center x-coordinate
		y1: center y-coordinate
		r1: x radius
		r2: y radius
		color: sets the line color
		fill: sets the fill color

Table 2
ASPECT syntax (part 2). The numerical values (coordinates) must be within the range 000-999 as required also by the TikZ language.

changes. These dependencies are due to the fact that we initially wanted to develop a quick-to-use tool capable, with a single command-line invocation, of generating from an ASP program a vector graphic representation of its solution(s) in both LaTeX and Portable Document Format (PDF) file formats at the same time.

In order to more easily handle different types of problems, four operating modes have been provided.

2.1. Standard Mode

The usage of ASPECT in standard mode is very simple. The interpreter should be invoked from the command line using the following scheme:

```
java ASPect <clingo arguments> <input file(s).lp>
```

First, the ASP program along with the ASPECT code for the visualization are passed to clingo together with the opportune solver options (if needed). The problem ASP code and the ASPECT code for visualization can be in the same file or in separate files since clingo also accepts multiple files as input. The output from clingo is redirected to a thread that is responsible, for each answer set, for generating the corresponding LaTeX file.

Then, each LaTeX file containing the TikZ description of the vector graphics is automatically converted into a PDF file using the pdfTeX extension.

The architecture of the ASPECT interpreter standard mode is sketched in Algorithm 1. The ATOM2TIKZ function (line 7) simply rewrites each ASPECT atom into a corresponding TikZ instruction. Below an example of an ASPECT atom converted into a TikZ instruction:

```
aspect_fillrectangle(3,15,5,17,gray)  ----> (atom2TikZ)
(atom2TikZ) ---> \draw [fill=gray] (3,15) rectangle (5,17);
```

Algorithm 1 Sketch of ASPECT interpreter

```
1:  AnswerSets = ASP SOLVER(solver arguments and input file(s))
2:  for each answerSet ∈ AnswerSets do
3:      LaTeXOutput = new output LaTeX file
4:      write preamble on LaTeXOutput
5:      for each atom $a$ ∈ answerSet do
6:          if $a$ ∈ ASPECT language then
7:              tikz_istruction = ATOM2TIKZ($a$)
8:              write tikz_istruction on LaTeXOutput
9:      write end section on LaTeXOutput
10:     PDFOutput = PDFLATEX(LaTeXOutput)
```

2.2. Merge Mode and Free Mode

The *merge* and *free* modes are designed specifically for handling problems that admit more than one solution and also implement support for the LaTeXbeamer class.

The merge mode, which can be invoked using the syntax:

```
java ASPect merge resize<dim> <clingo arguments> <input file(s).lp>
```

groups all the graphical representations of answer sets into a single LaTeX beamer document and consequently a single PDF file. This mode is aimed at creating animations for presentations in LaTeX in fact if the problem admits multiple solutions each of these will be placed in a different beamer frame. In the case of optimization problems, with this mode, it is possible to show in a simple way the successive solutions of increasing quality found by the solver up to the optimal one.

Use of the merge mode is not recommended if the user wants to constantly monitor the output produced by the ASP solver, as the PDF file will not be produced until the solver has completed the computation. The merge mode accepts a resize parameter that uses the `resizebox` command of the LaTeX graphicx package to resize the produced graphic. The value of the `dim` parameter uses `em` (em quadrat) as the unit of measure, which is applied as both vertical and horizontal length.

Free mode works similarly to merge, but allows the user to customize beamer frames and the `tikzpicture` environment by entering custom commands through the incorporation of two files called *before.tex* and *after.tex*. A schematic of the architecture of the ASPECT interpreter when operating in free mode is depicted in Figure 1. Free mode, similarly to the merge mode, can be invoked using the command line by entering:

```
java ASPect free <clingo arguments> <input file(s).lp>
```

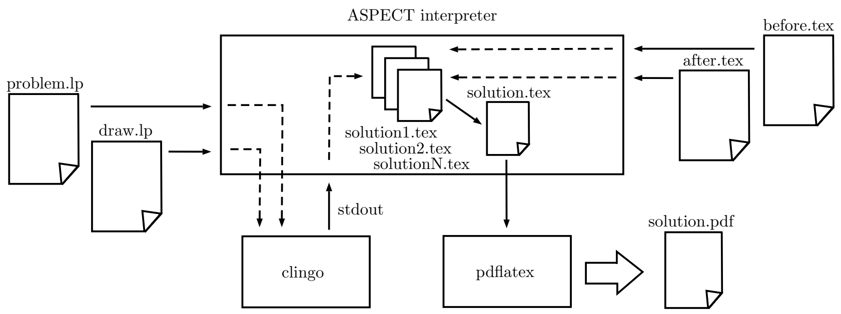


Figure 1: ASPECT interpreter architecture when working in free mode.

2.3. Graph Mode

Graph mode as the name suggests allows for quick visualization of solutions that can be represented by a graph. An example is shown in Section 3.1. This mode can be invoked with the command:

```
java ASPect graph <clingo arguments> <input file(s).lp>
```

Graph mode allows the user to insert graphical elements without having to worry about specifying coordinates, which will be handled automatically by the TikZ package.

Graph mode handles a different set of ASPECT atoms than the previous modes. The complete list of ASPECT atoms that can be used in this mode is given in Table 3. Again note how these graph elements do not require coordinates to be specified for their visualization.

Graphic element	ASPECT Atom	Parameters

node	`aspect_drawnode(A)`
	`aspect_colornode(A,fill)`
		A: node name (label)
		fill: sets the fill color

simple edge	`aspect_drawline(A,B)`
	`aspect_quoteline(A,B,"text")`
		A: node name first endpoint
		B: node name second endpoint
		”text”: sets the edge label

arrow edge	`aspect_drawarrow(A,B)`
	`aspect_quotearrow(A,B,"text")`
		A: node name arrow tail
		B: node name arrow tip
		”text”: sets the edge label

Table 3
ASPECT syntax supported in graph mode.

3. Examples

In this section we show how to use ASPECT syntax in various problems typical of the ASP context.

3.1. Graph Coloring

Graph coloring is a well-known problem that is often used as an example to introduce answer set programming to students. The problem consists of assigning colors to vertices of a graph such that two adjacent vertices do not share the same color.

The ASP encoding of the problem consists of `node(N)` atoms to denote the nodes of the graph where `N` is an index that identifies the node. Also part of the encoding are `arc(A,B)` atoms indicating the edges of the graph and `colour(N,Color)` atoms indicating the colour associated with each vertex.

The coordinates of the various graph nodes in the visualization are not important therefore we used the graph mode described in the previous section. Below are the ASPECT lines of code for the visualisation of the problem shown in Figure 2.

```
aspect_colornode(X,Color):- color(X,Color).
aspect_drawline(A,B):- edge(A,B).
```

In the first line, the `aspect_colornode` command draws a node graph with `Colour` filling. In the second line, the `aspect_drawline` command draws the edges of the graph.

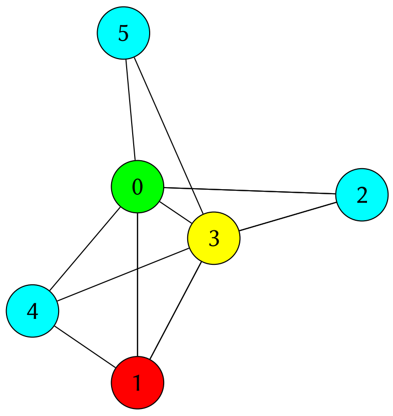


Figure 2: Graph coloring solution generated with ASPECT graph mode.

3.2. N-queens Problem

The N-queens problem is a classic puzzle that involves placing N chess queens on an $N \times N$ chessboard such that no two queens threaten each other: so, no pair of queens should share the same row, column, or diagonal. The challenge is to find a solution for any given value of N. Suppose that we are using an ASP encoding where the chessboard is described with an atom grid(I,J) for all the possible squares of the board, while the solution has an atom queen(R,C) indicating each queen positioned on the board; the programmer can draw the chessboard by adding just two lines of code.

```
aspect_fillrectangle(2*I-1,2*J-1,2*I+1,2*J+1,gray):- grid(I,J), I\2=J\2.
aspect_drawrectangle(2*I-1,2*J-1,2*I+1,2*J+1):- grid(I,J), I\2!=J\2.
```

These two lines of ASPECT code draw the squares of the checkerboard, each with side length 2 and centered in (2I,2J). In particular, the first line set the background color to gray for half of the squares to create the classic chequered pattern. In a similar fashion, we can draw the queens with:

```
aspect_imagenode(2*I,2*J,"queen.png",50):- queen(I,J).
```

ASPECT allows the programmer to further customize the visualization of a solution: as observable in the line above, users can include their own images in the plot just by specifying the respective filenames in the left-hand side of the desired rule. An example of the vector graphic produced by ASPECT, for this problem, using the syntax presented above and $N = 8$ is presented in Figure 3a.

On this well-known problem we also decided to evaluate the performance of the ASPECT interpreter. Table 4 shows how the time required by APSECT interpreter to generate the visualization varies as the problem size varies. The setup used in the tests consisted of an Intel® Core® i7-9750H CPU running at 2.6GHz with 16GB of RAM and Ubuntu 22.04 as OS.

N	clingo CPU Time [s]	TeX file writing time [s]	pdflatex compile time [s]	Total Time [s]
10	0.023	0.037	0.536	0.596
30	0.039	0.097	0.971	1.107
60	0.157	0.235	2.590	2.982
100	0.459	0.544	5.770	6.773

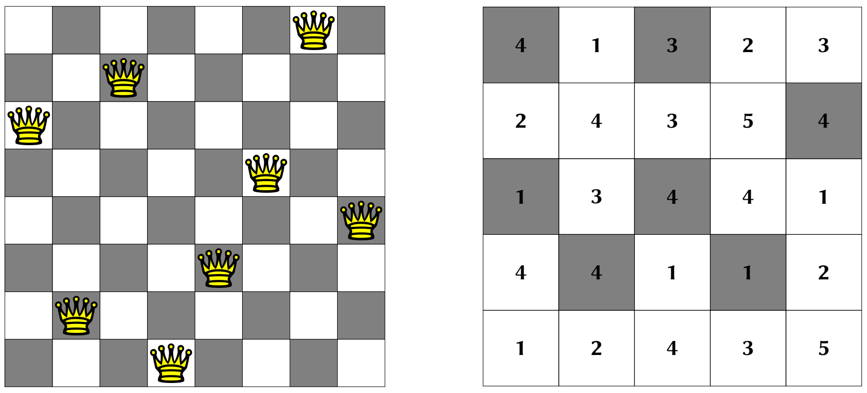
Table 4
Detailed time analysis of ASPECT interpreter (split into the various components). Times refer to solving the N-queens problem as the number N of chess queens varies. A visualization of the problem with $N = 100$ is shown in Figure 4.

3.3. Hitori

Hitori is a logic puzzle that is played on a grid of squares, where each square contains a number. The goal is to shade in some of the squares so that no number appears more than once in any row or column, and all unshaded squares are connected to each other horizontally or vertically.

Using a similar syntax as the one adopted for the *N-queens problem* to represent squares and nodes, we can obtain the graphical representation of the solution (see Figure 3b).

The squares are encoded using schema(X,Y,N) atoms where X,Y indicates the position of the square in the grid and N is the displayed number. The black(X,Y) atoms indicate the shaded squares.



(a) N-queens problem (b) Hitori

Figure 3: A board game and a logic puzzle generated with ASPECT.

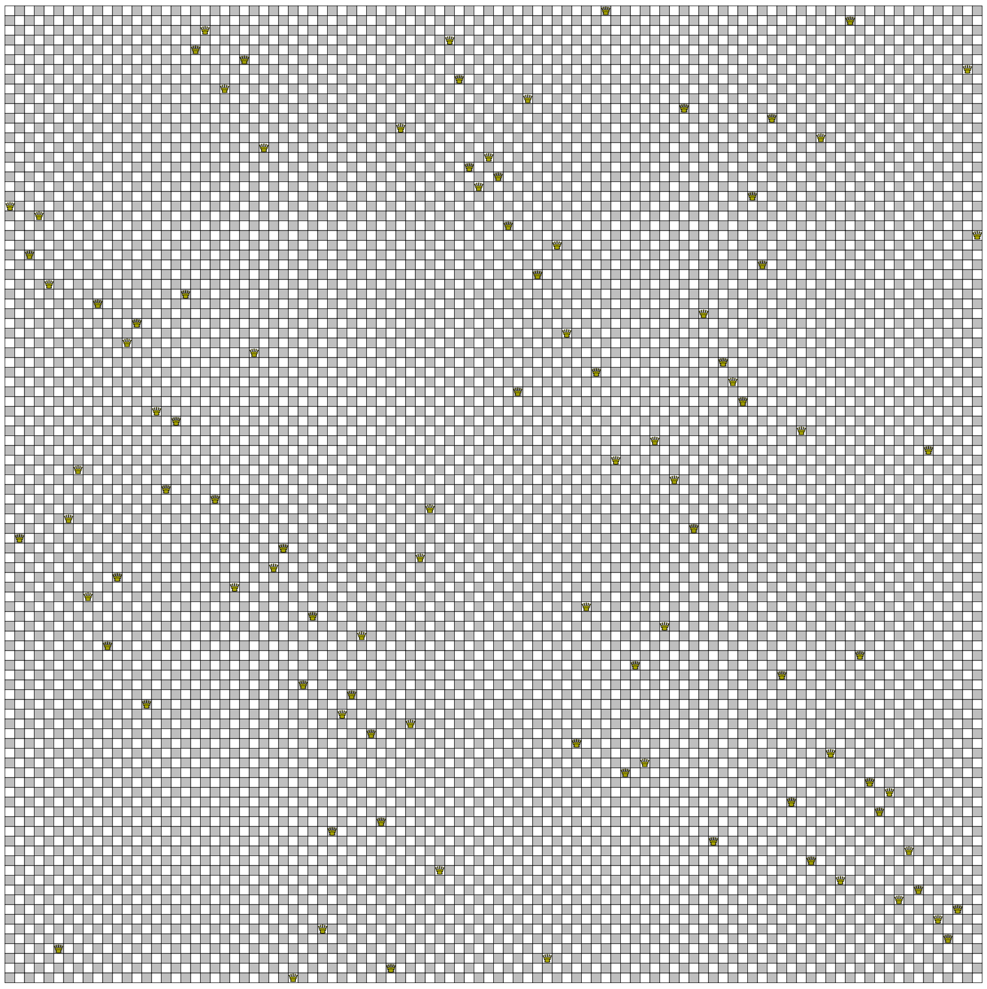


Figure 4: One solution of the N-queens problem with $N = 100$ chess queens. This image is generated using 10100 TikZ instructions.

```
aspect_fillrectangle(2*(X-1),2*(Y-1),2*X,2*Y,gray):- schema(X,Y,N), black(X,Y).
aspect_drawrectangle(2*(X-1),2*(Y-1),2*X,2*Y):- schema(X,Y,N).
aspect_drawnode(2*X-1,2*Y-1,N):- schema(X,Y,N).
```

Differently from the previous case, here “standard” nodes were needed in order to draw the correct number in each square, rendered through the variable `N`, but the programmer can arbitrarily choose to use any desired character or string to represent the nodes. Another famous logic puzzle that can be represented in a similar way to Hitori is Sudoku.

3.4. Minesweeper

Minesweeper is a logic puzzle that is played on a game board divided into cells. The player’s goal is to find where the hidden mines are located using clues found on the game board as numbers. The number in some cells corresponds to the number of mines in the 8 cells surrounding it.

A classic ASP coding for this game involves `number(X,Y,N)` atoms where `X` and `Y` are the row and column indexes of the game board and `N` is the clue that is displayed in the cell at that position. The `mine(X,Y)` atoms represent the position of the mines.

Using a combination of ASPECT atoms presented in the previous examples, the game board graphical representation can be easily obtained. In fact, this game requires the visualization of nodes both containing variables and containing images (for mines). An example is shown in Figure 5, and the code used to obtain the figure is shown below.

```
aspect_drawrectangle(2*(X-1),2*(Y-1),2*X,2*Y):- rows(X), cols(Y).
aspect_drawnode(2*X-1,2*Y-1,N):- number(X,Y,N).
aspect_imagenode(2*X-1,2*Y-1,"mine.png",40):- mine(X,Y).
```

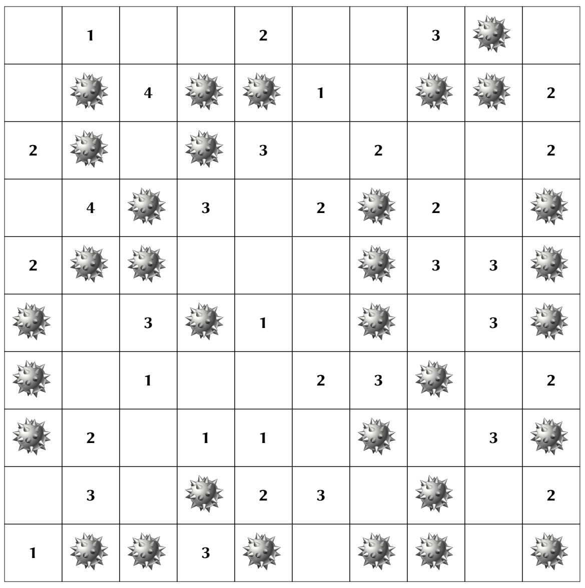


Figure 5: A minesweeper solved game board generated with ASPECT.

3.5. Scheduling

A scheduling problem involves determining the order of execution of a set of tasks while taking into account constraints such as: precedence constraints, capacity constraints, etc. The solution of a scheduling problem can be graphically represented by a Gantt chart. In the following we will show, with a simple example, how ASPECT can be used to generate Gantt charts. In our example we consider a very simple scheduling problem that consists of 4 activities each associated with its own duration and an identifying color: going to the bank (1 hour, green), going to the store (2 hours, blue), going to the post office (1 hour, red), and going to work (4 hours, yellow). The constraints of the problem state that it is necessary to go to the bank before the store and it is necessary to go to the post office before work. All activities should take place between 9 a.m. (`begin`) and 5 p.m. (`end`), and the activities obviously should not overlap.

The ASP encoding of the problem consists of `tasks(T,Colour)` atoms representing the various tasks to be scheduled where `T` is a sequential numerical identifier for each task and `Color` the colour associated with it. The `sequence(T,S,E)` atoms encode the solution; in these atoms `T` identifies a specific task while `S` and `E` are the start and end times of that task, respectively.

The complete ASPECT code with which we obtained the six solutions of the example problem described (see Figure 6) is shown below.

```
aspect_fillrectangle(S-begin,I,E-begin,I+1,Color) :- sequence(T,S,E), task(T,Color).
aspect_drawline(T-begin, 1, T-begin, N+1):- T=begin..end, n_tasks(N).
aspect_drawnode(T-begin, N+2, T):- T=begin..end, n_tasks(N).
```

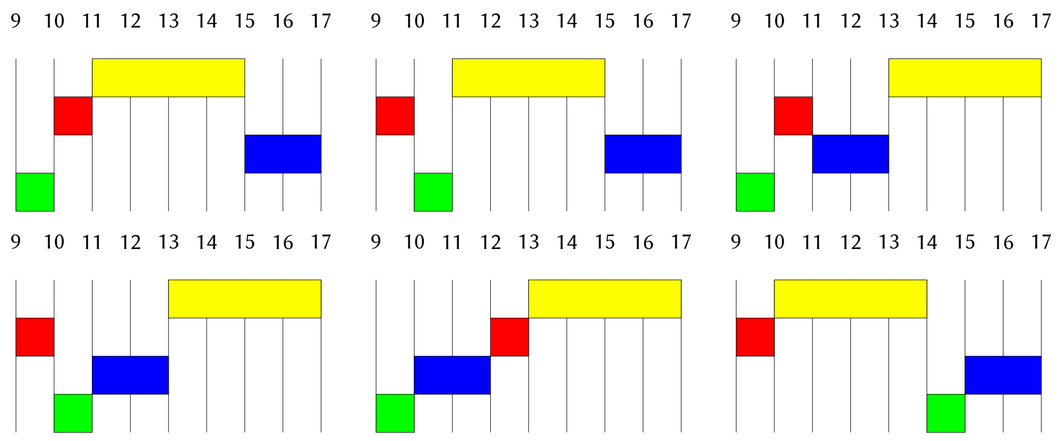


Figure 6: All six solutions of a simple scheduling problem generated by ASPECT.

3.6. Traveling Salesperson Problem

The Traveling Salesperson Problem (TSP) is a classic optimization problem that involves finding the shortest possible route that a traveling salesperson can take to visit a given set of cities exactly once and return to his starting point. The objective is to find the Hamiltonian cycle with the minimum total edge weight. Many instances of this problem include the coordinates of cities on the plane. Such coordinates in ASP can be represented by atoms as `point(C,X,Y)`. The solution is usually given as a predicate representing the arcs being followed: supposing that those arcs are identified by atoms `cycle(A,B)`, where A and B are the nodes of the graph with coordinates (X_A, Y_A) and (X_B, Y_B) respectively, we can obtain a graphic representation of (every) solution just by adding the lines:

```
aspect_drawline(XA,YA,XB,YB):- cycle(A,B), point(A,XA,YA), point(B,XB,YB).
aspect_fillellipse(XA,YA,1,1,gray):- point(_,XA,YA).
```

to the ASP code of the problem. The first rule will plot the forementioned arcs, while the second one is in charge of drawing the nodes between the lines, represented as gray filled circles.

As known, optimization problems usually find a plethora of feasible solutions while searching for the optimal one(s), i.e. the solution(s) optimizing the objective function: using ASPECT, programmers have the possibility of automatically plotting all the solutions found, either in separate files or in a single one. A visualization using APSECT of a TSP is shown in Figure 7, in particular, the first solution found by the ASP solver and the optimal solution are shown.

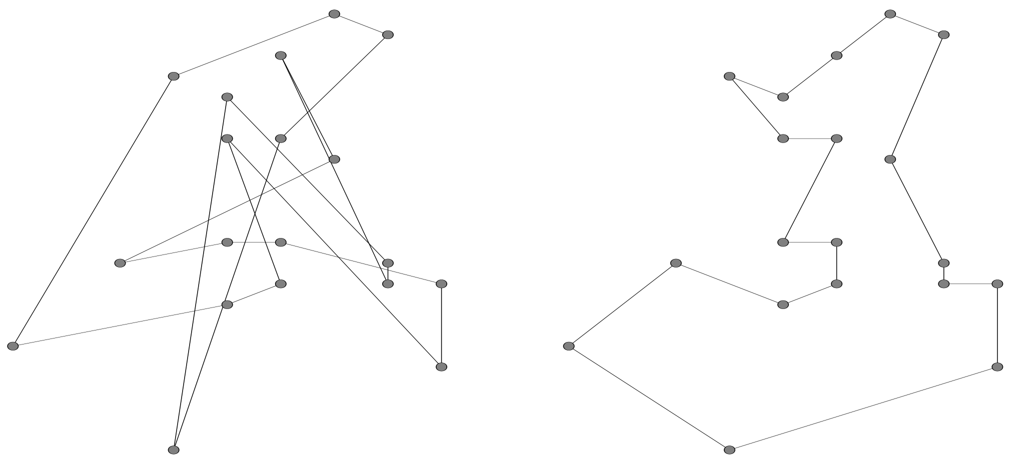

(a) First solution (b) Optimal solution

Figure 7: Two solution of a TSP produced by ASPECT.

4. Related Work

The difficulty in interpreting answer sets has led to the development over the years of tools, leveraging different technologies, aimed at producing easier-to-read graphical representations.

ASPViz [7] is a Java program that constructs images from the answer set of a given program. ASPViz uses the ASP language to define how elements of the problem solution should be displayed. Atoms of ASPViz language are extracted from the answer set and used to produce a graphical visualization using the Java SWT graphical toolkit. In addition to rendering answer sets of a program individually, ASPViz may be used also to create animations and multi-framed image visualisations.

Although ASPViz offers powerful visualization options like the animation of multiple solution outputs and the possibility to customize the atoms with external image files, it forces the user to create a second `.aspviz` program. In that file, the user has to specify the relationships between the atoms of the original ASP program and the desired visual representation, together with the customization options: such process could be cumbersome in some applications, we find that our approach is simpler for the novice user.

IDPDraw [6] is a tool for visualizing finite structures that can be used to visualize the output of an ASP solver written in C++.IDPDraw recognizes 8 types of atoms that allow the programmer to draw elementary geometric shapes (e.g., polygons, ellipses) and define their display properties (e.g. foreground color, background color, position, etc). IDPDraw atoms are interpreted and displayed through a user interface based on the Qt library. As in the case of the previous system, it is possible to create animations by associating each atom with a time argument. ASPIDE [9], an Integrated Development Environment for ASP, includes IDPDraw as visualization tool. The sources of IDPDraw are available online[3], but they rely on outdated Qt4 libraries which, having reached the end of life, have been removed from the repositories of the major Linux distributions.

Kara [5] is a tool created following the approach already introduced by ASPVIZ and IDPDraw as it uses the ASP language itself to define the visualization of answer sets. Unlike the latter, which position graphic primitives according to static coordinates only, Kara allows for more high-level specifications, supporting graph structures, grids, and relative positioning of graphical elements. Kara is written in Java and integrated in the SeaLion [10] integrated development environment for ASP. The SeaLion executable we obtained via the link[4] reported in [10] has as its latest update May 2017 and we were not able to run it on the latest Ubuntu 22.04.

More recently Dovier et al. [11] developed, for teaching purposes, a Java tool capable of visualizing the output returned by the ASP solver `clingo` [3]. However, this tool turns out to be limited to displaying only problems concerning a grid (e.g. magic square, sokoban, Sam Lloyd's puzzles etc.) which are the ones considered in their publication.

A recent publication from last year concerning the visualisation of answer sets presented `clingraph`. Clingraph [8] is a tool which aims at visualizing answer sets by means of ASP language itself. Clingraph is based on ASPViz, which, however, has been completely redesigned and adapted to work with modern ASP systems. Clingraph was originally designed as a visualizer for graphs defined as a set of facts but is now able to generate images similar to those produced by ASPECT. Among other features, it allows the export of images in LaTeX code but lacks the more advanced features provided by ASPECT such as integration with beamer for the automatic creation of presentations and the possibility of customising LaTeX document by importing external files.

5. Conclusion

We presented ASPECT, a language for describing graphical elements that can be easily provided as answer sets, letting the user describe complex drawings in an intuitive and declarative way. The ASPECT syntax consists of special atoms that define the graphical rendering of geometric

[3]https://wms.cs.kuleuven.be/dtai/pages/software/idpdraw/idpdraw
[4]https://sourceforge.net/projects/mmdasp/

Tool Name	**Language**	**Year of Publication**	**User interface**	**Standalone software ?**	**Exportable file formats**
ASPViz [7]	Java SWT	2008	command line	Yes	SVG
IDPDraw [6]	C++ Qt	2009	graphical	Yes	-
Kara [5]	Java	2011	graphical	No (plugin for SeaLion IDE)	SVG
Dovier et al. [11]	Java	2016	graphical	Yes	-
clingraph [8]	Python	2022	command line	Yes	JPEG, PNG, GIF, SVG, LaTeX code
ASPECT	Java	2023	command line	Yes	LaTeX code (w/ Beamer Integration)

Table 5
Comparison of different tools for graphical representation of answer sets.

primitives such as points, lines, polygons, ellipses, etc. Users can combine ASPECT atoms to generate figures of any complexity as answer sets.

We also presented a preliminary version of the ASPECT interpreter, written in Java, that is responsible for converting ASPECT atoms into the popular LaTeX markup language to produce vector graphics. A preliminary version of the interpreter for ASPECT is available online at https://github.com/abertagnon/aspect. Since it uses LaTeX as output format, the documents produced by ASPECT are easy to embed in scientific articles, course handouts, and presentations. Nonetheless, the LaTeX output can also be converted to other (vectorial) graphic formats for other uses, such as in web pages.

The syntax of ASPECT is inspired by the TikZ language. To date, the graphical primitives implemented in ASPECT are limited compared to those provided by TikZ especially in terms of the style properties of geometric primitives. However, the structure of the interpreter is easily adaptable to allow extension to more TikZ features in the future.

A future extension is to make the ASPECT interpreter independent from `clingo` and the pdfTeX extension. The choice to introduce these systems into the pipeline while speeding up image generation at the same time prevents the user from choosing their favorite ASP solver and LaTeX compiler.

Another extension would involve visualizing a single answer set through a sequence of graphical representations, for problems whose answer set is a sequence of actions. This would be useful in planning problems (e.g. Tower of Hanoi, Block world) where atoms include the timestep at which the action they represent is actually performed.

References

[1] O. Garcia, R. Perez, B. Silverman, H. Austin, R. Baum, L. Brady, R. Cameron, S. Castaneda, J. Chen, P. Dey, G. DiCristina, A. Elmaghraby, R. Foster, C. Freeman, M. Kirch, A. Lawrence, A. Manesh, S. Manickam, C. Ramamoorthy, R. Rariden, U. Reichenbach, S. Rosenbaum, F. Saner, F. Severance, C. Torsone, D. Valentine, H. Van Landingham, R. Vasquez, On

teaching AI and expert systems courses, IEEE Transactions on Education 36 (1993) 193–197. doi:10.1109/13.204845.

[2] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. A. Kowalski, K. A. Bowen (Eds.), Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes), MIT Press, 1988, pp. 1070–1080.

[3] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo = ASP + control: Preliminary report, CoRR abs/1405.3694 (2014).

[4] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Logic 7 (2006) 499–562. URL: https://doi.org/10.1145/1149114.1149117. doi:10.1145/1149114.1149117.

[5] C. Kloimüllner, J. Oetsch, J. Pührer, H. Tompits, Kara: A system for visualising and visual editing of interpretations for answer-set programs, in: H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, A. Wolf (Eds.), Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers, volume 7773 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 325–344. URL: https://doi.org/10.1007/978-3-642-41524-1_20. doi:10.1007/978-3-642-41524-1_20.

[6] J. Wittocx, Idpdraw, a tool used for visualizing answer sets, 2009.

[7] O. Cliffe, M. D. Vos, M. Brain, J. A. Padget, ASPVIZ: declarative visualisation and animation using answer set programming, in: M. G. de la Banda, E. Pontelli (Eds.), Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings, volume 5366 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 724–728. URL: https://doi.org/10.1007/978-3-540-89982-2_65. doi:10.1007/978-3-540-89982-2_65.

[8] S. Hahn, O. Sabuncu, T. Schaub, T. Stolzmann, Clingraph: ASP-based visualization, in: G. Gottlob, D. Inclezan, M. Maratea (Eds.), Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 401–414. URL: https://doi.org/10.1007/978-3-031-15707-3_31. doi:10.1007/978-3-031-15707-3_31.

[9] O. Febbraro, K. Reale, F. Ricca, ASPIDE: integrated development environment for answer set programming, in: J. P. Delgrande, W. Faber (Eds.), Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings, volume 6645 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 317–330. URL: https://doi.org/10.1007/978-3-642-20895-9_37. doi:10.1007/978-3-642-20895-9_37.

[10] J. Oetsch, J. Pührer, H. Tompits, The sealion has landed: An IDE for answer-set programming - preliminary report, in: H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, A. Wolf (Eds.), Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers, volume 7773 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 305–324. URL: https://doi.org/10.1007/978-3-642-41524-1_19. doi:10.1007/978-3-642-41524-1_19.

[11] A. Dovier, P. Benoli, M. C. Brocato, L. Dereani, F. Tabacco, Reasoning in high schools: Do it with ASP!, in: C. Fiorentini, A. Momigliano (Eds.), Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016, volume 1645 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 205–213. URL: https://ceur-ws.org/Vol-1645/paper_9.pdf.