FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

ARVis - User Guide

Visualization of relationships between answer sets

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Andreas Jusits
Matrikelnummer 0928977

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Privatdoz. Dipl.-Ing. Dr.techn. Stefan Woltran
Mitwirkung: Univ.Ass. Dipl.-Ing. Günther Charwat

Wien, 17.07.2012

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

ARVis - User Guide

Visualization of relationships between answer sets

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Andreas Jusits
Registration Number 0928977

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Privatdoz. Dipl.-Ing. Dr.techn. Stefan Woltran
Assistance: Univ.Ass. Dipl.-Ing. Günther Charwat

Vienna, 17.07.2012

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Andreas Jusits
1220 Wien, Österreich

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

Answer set programming is concerned with the computation of models, also called answer sets, that represent a solution to a problem. Usually such problems are search problems, like finding a solution to a given Sudoku puzzle. These solutions are computed with the aid of answer set solvers. Because it requires much effort to interpret answer sets after execution of a program with an answer set solver, software-tools like ASPViz, IDPDraw, Kara or ASPIDE support a graphical visualization for the computed answer sets. The innovation of the software-tool ARVis developed in the course of this work, consists of visualizing the relations between answer sets. It allows the calculation of answer sets in a first step and the computation of the relations between these answer sets in a second step. This bachelor thesis describes the background mechanism (all necessary steps, possible inputs and computation algorithms) for visualizing the relations between answer sets. ARVis is implemented as a wizard that is divided into five steps. The key content of this bachelor-thesis is the User-Guide for ARVis that describes the functionalities of the tool on the one hand and illustrates, by means of an example that continues through all visualization-steps, how to achieve a visualization of the relations between answer sets on the other hand.

Kurzfassung

Answer-Set Programmierung (ASP) beschäftigt sich mit der Berechnung von Modellen (auch Answer-Sets genannt), die eine Lösung zu einem Problem repräsentieren. Meist sind diese Probleme Such-Probleme, wie beispielsweise das Lösen eines Sudoku-Rätsels. Die Berechnung von Modellen erfolgt anhand von AS-Solvern. Da die Visualisierung von Answer-Sets die Interpretation der Modelle erleichtert, gibt es bereits einige Software-Tools wie ASPViz, IDPDraw, Kara oder ASPIDE, die eine graphische Visualisierung der berechneten Answer-Sets unterstützen. Die Innovation des Software-Tools ARVis, welches im Zuge dieser Bachelorarbeit entwickelt wurde, besteht darin, Relationen zwischen Answer-Sets zu visualisieren. In einem ersten Schritt erfolgt die Berechnung von Modellen, welche weiters in einem zweiten Schritt zueinander in Beziehung gestellt bzw. diese Beziehungen durch einen Graphen illustriert werden. In dieser Bachelorarbeit wird anhand von zusammenhängenden Beispielen der Hintergrundmechanismus (alle Schritte, Eingabemöglichkeiten, Einschränkungen und Berechnungsalgorithmen) erläutert, um schlussendlich zu einer Visualisierung der Beziehungen zwischen Answer-Sets zu gelangen. Der vollständige Vorgang erfolgt in ARVis mit Hilfe eines Wizards, der aus fünf Schritten besteht. Kern dieser Bachelorarbeit ist das Benutzerhandbuch von ARVis, das einerseits die Funktionalitäten des Tools aufzählt bzw. beschreibt und andererseits wiederum anhand eines Beispiels die Schritte näher erläutert, um zur Visualisierung der Beziehungen zwischen den Answer-Sets zu gelangen.

Contents

CHAPTER 1

Introduction

Answer Set Programming (ASP) defines a declarative problem solving paradigm. On the basis of a search problem and an instance, an asp-solver computes all intended models to obtain all solutions (answer sets) for the search problem. Contrary to the existing idea of visualizing answer sets we found ways of visualizing the relations between the models.

This bachelor thesis is concerned with a user-side guide for the visualization tool ARVis which was developed in the course of this work. ARVis allows the computation of answer sets in a first step and the calculation of the relations between these answer sets in a second step. The end result of the depicted relations is a graph with nodes (represent answer sets) and edges (represent relations). This standalone-tool is characterized by versatility (e.g. Windows, Linux, Mac OS, ...) and the covering of many applications (e.g. Traveling Salesman, Sudoku, ...). After launching ARVis a user-interface appears that contains a wizard divided into five steps to achieve a visualization of answer sets. The innovation of this software lies in the mechanism of visualizing relations that we will present in the following chapters.

In Chapter 2 we will refer to related works to show that on the one hand answer set visualization in general is not an innovative idea from us, but on the other hand the visualization of the relations between answer sets is a completely new mechanism that is founded in this bachelor thesis. We will give a short summary on existing visualization concepts and tools. Futhermore, the differences between these tools are demonstrated.

Chapter 3 describes our approach for visualization, building the background mechanism of the tool. This mechanism is divided into four steps that are developed in course of this bachelor thesis. The basic steps are demonstrated by means of an example that continues through all four visualization-steps. The intention of this chapter is to improve the understanding of ARVis on the one hand, and give the user the chance to develop similar tools based on the idea of visualizing relations between answer sets on the other hand.

The heart of this bachelor thesis is accurately described in Chapter 4 that contains a user-guide for the tool. By the help of a detailed example and screen-shots the functionality and flexibility of ARVis is demonstrated. Each wizard-step is equivalent to a step described in the Chapter 3. One additional step is introduced (i.e. edge/highlighting-selection). This step is skippded in Chapter 3 as it is not essential for understanding the general concept. Besides the base-functionalities to get a graph result, additional-features like the configuration environment where one can set up parameters for an arbitrary asp-solver are shown. Furthermore, the data export feature and possibilities for editing the graph output visualization are presented.

In Chapter 5 a conclusion is given. It also contains a short summary of the existing tool and an outlook on possible future work. Furthermore, possible enhancements for ARVis are listed and discussed.

CHAPTER 2

Answer Set Visualization - State of the Art

This chapter gives an overview about existing answer set visualization tools. Furthermore we summarize the features and differences of the tools ASPViz, IDPDraw, Kara and ASPIDE.

It requires much effort to interpret the output (answer sets) after execution of a program with an as-solver. Answer set visualization tools serve the purpose of better understanding the result, i.e. the output. Furthermore some tools offer possiblities to edit the output or to postprocess the output with further tools.

2.1 ASPViz

ASPViz [3] is a Java stand-alone tool for answer set visualization of a given answer set program. It uses a Java-native graphic toolkit (SWT - Standard Widget Toolkit) for the graphical display (construction of two-dimensional images). It is possible to visualize answer sets in an arbitrary way (e.g.: as rectange, text, circle, ...) on the basis of predefined predicates. The visualization tool takes two programs that are processed with an as-solver for visualization:

1. Answer set base program: Contains a given problem, e.g.: Sudoku-problem.

2. Visualization program: Consists of visualization-predicates that define how the result should be visualized. Furthermore it combines the necessary literals for the graphic-visualization with the output of the answer set base program. Possible literals for graphic-visualization could be the definition of brushes, text styles, grid-look, cell-look,...

After solving the two programs an arbitrary amount of answer sets is computed. ASPViz takes the predicates from the answer set and produces a graphic using SWT. ASPViz builds the associated native objects (in SWT) by means of the visualization-predicates defined in the visualization program (e.g.: brushes, text style, fonts, ...). After the object generation, each associated base graphic element, using the native drawing function, is drawn for the visualization output.

2.2 IDPDraw

IDPDraw [7] is also a tool to visualize answer sets. It is a stand-alone tool and written in C++ (depends on qt-libraries). One has to choose a solver that computes the result. IDPDraw consists of a predefined vocabulary (predicates) that has to be specified by the user to visualize the result.

IDPDraw ingores all predicates from the result, except the predefined atoms that define how the input should be visualized. An example for a predefined atom would be the atom `idpd_polygon(n, Name1,..., Namen, x1, y1, x2, y2,..., xn, yn)` that creates a polygon element with an arbitrary amount of vertices that are connected by lines.

IDPDraw is a simple tool for answer set visualization. A specific feature of this tool is the support of timepoints. Answer sets may have a different structure at different timepoints. With this feature it is possible to reconstruct the evolution of answer set solving. So IDPDraw makes use of a list of predefined predicates (e.g. `idpd_polygon`) plus the same predicates with the ending '_t' and an additional parameter for the timepoint-value. It is possible to go forward and backward from and to each timepoint-definition.

2.3 Kara

Kara [6] is part of SeaLion (Eclipse-based IDE for ASP) and an improvement compared to the tools described above. It is written in Java. Kara uses the language of answer set programming for the visualization of interpretations. Kara supports graph structures, grids and relative positioning of graphical elements. Another additional feature of Kara in contrast to the other tools is that visualizations are graphically editable. That means that it is possible to change sizes, delete elements, insert elements, ... Kara supports a vector graphic export where models can be stored and edited externally.

The following features are specific for Kara:

- Grid structure: Two-dimensional grids may be generated. To define a grid the number of rows and columns can be set. Furthermore it is possible to fill the cells of the grid with (predefined) values that can be edited.
- Visual editing: By specifying predefined atoms the target of visual editing is achieved. Elements for deleting, creating, changing and hiding elements can be defined. These

operations can be applied to elements in the GUI-Editor.

- Relative positioning: In order to achieve a fixed positionining, predefined predicates need to be specified. Otherwise the positioning occurs automatically. The user can also set the position of an element relatively to another element. It is furthermore possible to define the depth-level of an element (i.e. which element overlaps the other).

- Graph support: In order to visualize answer sets in a graph, predefined predicates are used. It is not only possible to define a graph, one can also define the form and size of a node. Moreover one may define the ellipse as node of the graph and the edges between the nodes. Furthermore the labels of the nodes and the backround color can be defined.

2.4 ASPIDE

ASPIDE [5] is an extensive IDE for answer set programming. It integrates tools for adaptations (autocompletion, code-templates,...) and tools for visualizations (debugging, profiling, ...). The features of ASPIDE are comparable with the functionalities of other IDEs like Eclipse. They provide functionalities like workspace management, an advanced text editor (e.g. autocompletion of predicates), code checking, error handling, debugging, profiling, execution configuration, presentation of results, interaction with databases, ... The component in ASPIDE that regards to the feature of visualizing answer sets is the Visual Editor.

The Visual Editor allows the drawing of as-programs instead of writing them as text files. It consists of a graphic environment for CRUD-operations (Create, Remove, Update, Delete) that have influence on the as-program. The effort of learning the syntax of asp can be spared. The Visual Editor supports the asp system DLV and the storing and loading of as-programs. Furthermore it is possible to switch between the visual and the textual representation of the as-program.

2.5 Comparison between the Visualization-Tools

Table 2.1 gives an overview of the features provided by ASPViz, IDPDraw, Kara and ASPIDE.

Each tool supports the main feature of visualizing answer sets. ASPViz and IDPDraw restrict the functionalities on this main feature and the feature of creating animations. The latter is a specific functionality that argues for ASPViz and IDPDraw. In contrast Kara and ASPIDE provide further features like editing or debugging. Therefore these two tools are more professional. Kara offers the most options for visualization. A unique feature of ASPIDE is the ability to draw as-programs instead of writing them as text files.

Figure 2.1 shows an example visualization of ASPViz: A maze is drawn where the mouse has to find its way out of the labyrinth. It illustrates the additional feature 'Creation of animations' as listed in Table 2.1.

Table 2.1: Feature-Comparison of Visualization Tools

Feature	ASPViz	IDPDraw	Kara	ASPIDE
Relative positioning	NO	NO	YES	YES
Support of automatic layouting of graphs or grid structures	NO	NO	YES	YES
Debugging	NO	NO	NO	YES
Profiling	NO	NO	NO	YES
Test suite	NO	NO	NO	YES
Configuration of the execution	NO	NO	NO	YES
Implementation of a possibility to determine the depth-level of an element	NO	YES	YES	YES
Support of vector-graphics-export (in SVG format)	YES	NO	YES	NO
Creation of animations	YES	YES	NO	NO

Figure 2.2 represents a Tangram puzzle computed in IDPDraw. Tangram is a puzzle consisting of pieces from a square (triangles, parallelograms, ...). As one can see in the figure the 'Creation of animations'-feature is supported by the Navigation-Buttons (Next, Previous, ...). In the case of Tangram one can see each step of solving the puzzle.

Figure 2.3 illustrates Kara, a plugin for an Eclipse-based IDE (SeaLion) for answer set programming. It shows a graph that is constructed by Kara. It graphically represents the relations (edges) between predicates (nodes). Furthermore one can see that this visualization is editable.

Figure 2.4 represents the Visual Editor of ASPIDE. It shows the visual representation of an as-program that is in the process of creation. The graph illustrates the definition of predicates (and the relations between them) for the as-programs without using a text editor (no knowledge of the syntax neccessary).

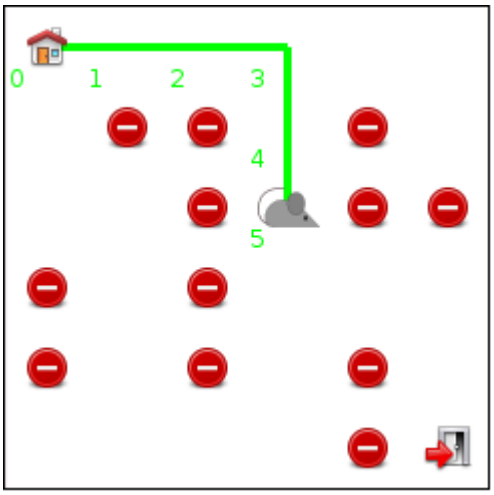

Figure 2.1: ASPViz [8]

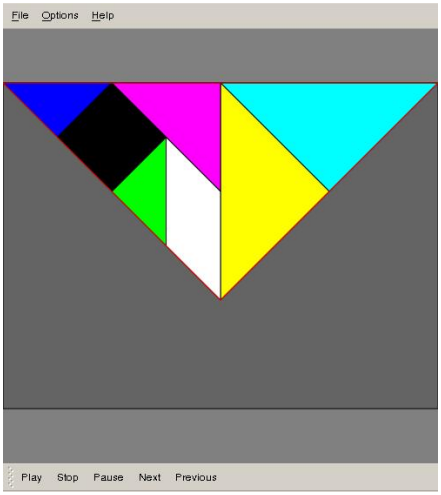

Figure 2.2: IDPDraw [7]

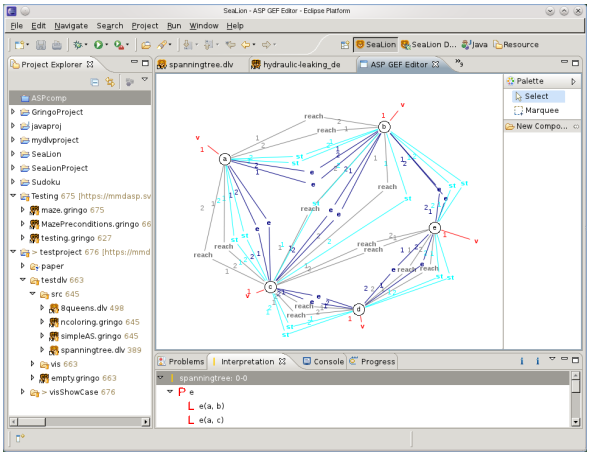

Figure 2.3: Kara [6]

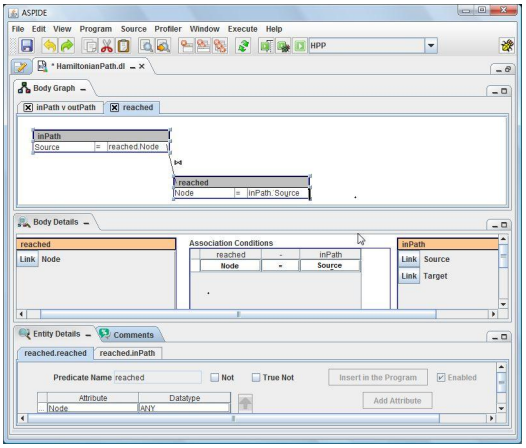

Figure 2.4: Visual Editor of ASPIDE [2]

CHAPTER 3

Four Steps to Visualization

3.1 Overview

The general goal of our new system is to represent the relations among answer sets. To achieve the target of visualizing answer sets we define four general steps. Figure 3.1 illustrates these four steps.

1. Solve the Base-Program: The first step is defined by the general use of answer set solving. An answer set solver is used for computing models (answer sets) of a user specified problem. A problem consists of the input instance (some kind of knowledge base) and rules that define the problem declaratively.

2. Generate new Input Instance: To represent the relations between the generated answer sets from step one we need a step that converts the answer sets into a single input instance. The second step defines a core-action for the visualization of answer sets. The main function is to generate ground literals out of the answer sets by a defined algorithm in order to prepare the new input instance for the next steps.

3. Compute Relations between Answer Sets: In the third step an answer set solver is called again to apply an answer set program to the new input instance. The resulting models provide the basis for the visualization. The user can specify any program that constructs relations between the models.

4. Generate Graph: The functionality of the last step deals with generating a graph. This graph is built up of nodes and edges. Each node of the graph represents an answer set that was computed in step one. These answer sets of the generated model created in step three defines the relationships (unidirectional edges) between the nodes.

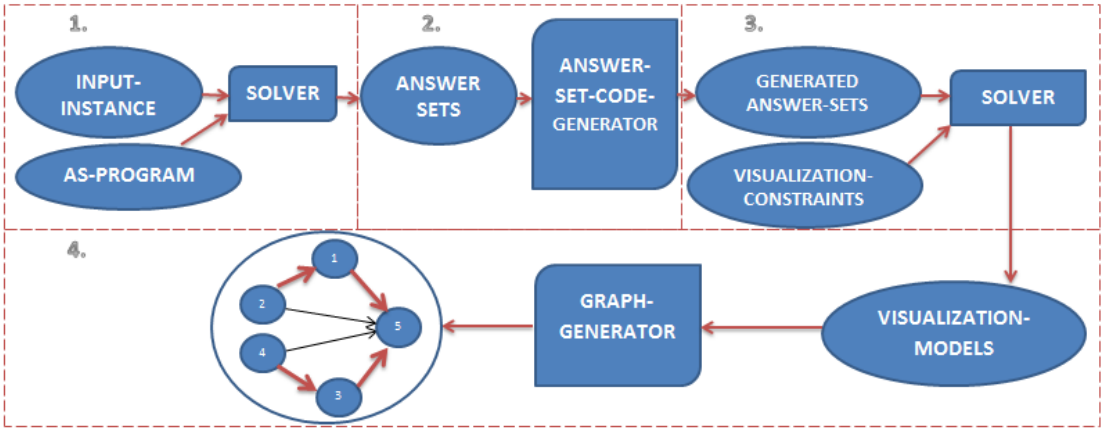

Figure 3.1: Overview over the 4 visualization steps

To give a better understanding about the application of the four steps we describe each step by means of a continuous (through all four steps) example. It is not relevant to question the meaning of the following example because it is only used to illustrate the four steps. It is chosen because it builds a potential application area for the visualization-mechanism.

Example. As running example we use a problem that stems from the field of abstract argumentation. In abstract argumentation we consider arguments and relations between them. This can be illustrated as a graph, where vertices represent arguments and edges represent attack-relations. The most popular formulization of such argumentation frameworks was introduced by Dung in 1995 [4]. We consider admissible semantics which asks for the sets of arguments that are:

1. conflict-free: No adjacent arguments are selected together.
2. every selected argument is defended against attacks it receives: Attackers of selected arguments are again attacked by the set of selected arguments.

3.2 Step 1: Solve the Base-Program

The user provides the following encodings as input:

1. Input instance (knowledge base): Only consists of ground literals (terms are no variables)
2. AS-Program (rules): Used to query a solution (answer sets) out of the knowledge base. The constraints are called rules. A program is then a finite set of rules. In practice you will have disjunctive rules that define the search space and integrity constraints that remove illegal branches.

These two parts should be defined by the user.

```
Example - Input-Instance:

%arg(x)     ... represents arguments(or vertices) in an
     argumentation framework(AF)
%att(X, Y) ... represents attacks(or edges) in the AF

arg(a). arg(b). arg(c). arg(d).
att(a,b). att(b,a). att(a,c). att(b,c). att(c,d).
```

Figure 3.2 shows the input-instance of our running example.

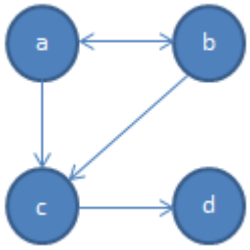

Figure 3.2: Example - Input instance

```
Example - AS-Program (rules):

% guess a solution candidate for each argument X,  if X is in the set
% of selected arguments
in(X) v out(X) :- arg(X).

% delete the sets where we have a conflict
:- att(X,Y), in(X), in(Y).

% delete all sets where a selected argument is attacked by any other argument
% that is not attacked by the set.
:- out(X), in(Y), att(X, Y), not def(X).

% compute if arguments that are not selected are attacked by the set.
def(X) :- out(X), in(Y), att(Y, X).
```

To solve the program we have to use an answer set solver. The encoding of our running example is specified in dlv syntax. Model(s) represent the output of the solver - our answer sets. We need the answer sets as fundament for the next step (Step 2: Generate new Input Instance). A visualization of the solver-output from the example can be seen in Figure 3.3.

```
Example - Solver output:

% only predicates in and out are used
ID 1: out(a) out(c) out(d) in(b)
ID 2: out(a) out(c) in(b) in(d)
ID 3: in(a) out(b) out(c) out(d)
ID 4: in(a) in(d) out(b) out(c)
ID 5: out(a) out(b) out(c) out(d)
```

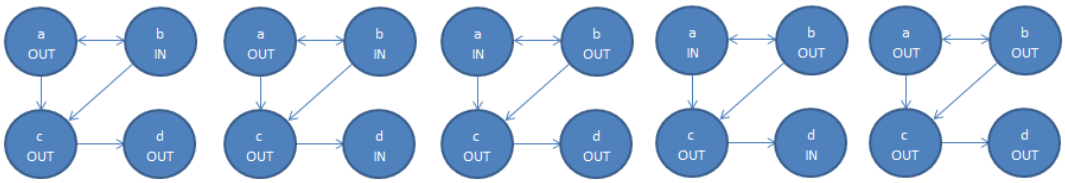

Figure 3.3: Example - Answer sets after ASP-Solving

3.3 Step 2: Generate new Input Instance

Based on the computed answer sets, we need a new input instance for the visualization of dependencies between the different models. The approach to generate new ground literals out of one answer set is described as follows:

Consider the following answer set that was computed by the previous step:

```
ID as_number: a(a1, a2, ..., an), b(b1, b2, ..., bm), ...
```

The ASP-Code-Generator then generates the following output:

```
as(as_number, a, a1, a2, ...., an),
as(as_number, b, b1, b2, ..., bm), ...
```

This code-generation is executed for each answer set. What we have is a mapping from all answer sets to a single answer set where each gets assigned to the number from the original answer set. This id starts with 1 and is incremented by 1. This standard-algorithm is defined to ensure that the input instance for the next step (Step 3: Compute Relations between Answer Sets) always has the same structure.

```
Example - Generated ASP-Code:

as(1, out, a). as(1, out, c). as(1, out, d). as(1, in, b).
as(2, out, a). as(2, out, c). as(2, in, b). as(2, in, d).
as(3, in, a). as(3, out, b). as(3, out, c). as(3, out, d).
as(4, in, a). as(4, in, d). as(4, out, b). as(4, out, c).
as(5, out, a). as(5, out, b). as(5, out, c). as(5, out, d).
```

3.4 Step 3: Compute Relations between Answer Sets

The target of this step is to obtain relations between the answer sets. The user is free to highlight edges with a user-defined characteristic. The difference to the first step (Step 1: Solve the Base-Program) is that the input instance is the generated code from the last step (Step 2: Generate new Input Instance).

In addition we also need to apply rules (constraints) to the generated input instance. They define the dependencies between the answer sets (represented by their id). The dependencies could be shown in the following two ways, where predicates must be selected that describe the relations between the answer sets:

1. Edges: These edges represent the relations between answer sets.
2. Highlight-Edges: In addition to show relations between answer sets, it is also possible to highlight specific edges.

```
Example - Visualization-Constraints:

% In this example we are interested in the maximal answer sets
% wrt. set inclusion. We draw an edge(X,Y) if the arguments A in X,
% colored with in (i.e. in(A)), is a superset of the arguments in Y.

% Computes if some argument is in answer set X but not in answer set Y
argNotContained(X,Y) :- as(X, in, A), as(Y, out, A).

% We draw an edge if X is a superset of Y,
%i.e. if there is some argument in X but not in Y.
edge(X,Y) :- as(X, _, _), as(Y, _, _), argNotContained(X, Y),
                    not argNotContained(Y, X).
% Compute all transitive edges.
% transitivity: for ALL a, b, c E X : (a R b) AND (b R c) => a R c
transitiveEdge(X,Z) :- edge(X,Z), edge(X, Y), edge(Y, Z).

% Compute all edges for all answer sets that are not transitively reachable
edgeVis(X,Y) :- edge(X,Y), not transitiveEdge(X,Y).
%edge(X, Y) ... edges,  edgeVis(X, Y) ... highlight-edges
```

Now we solve the program (e.g. with dlv) again. The output can contain one or more visualization-models. Any predicate with arity 2 can be chosen as edge or highlight-edge. The two parameters have to refer to the `as_number` of answer sets from step 2.

```
edge_name(from_node, to_node).
edge_highlight_name(from_node, to_node).
```

The first term defines the beginning and the second term the ending of the unidirectional relation. The error-case that one parameter is not a valid number has to be taken into account.

```
1 Example - Visualization-output:
2
3 edge(1,5). edge(2,1). edge(2,5). edge(3,5). edge(4,3). edge(4,5).
4 edgeVis(1,5). edgeVis(2,1). edgeVis(3,5). edgeVis(4,3).
```

3.5 Step 4: Generate Graph

A graph is used for the visualization of the relations between answer sets. Each node represents an answer set (with an unique id) and each relation is shown by an edge. A special characteristic between two nodes can be expressed by highlighting the edge between them. The following algorithms can be applied in order to compute a visualization-graph out of the visualization output (answer sets). There are two possibilities to generate a graph from answer sets:

1. Generate 1 graph: Loop all edges from each answer set. Remember the terms to get a distinct list of all nodes.

   ```
   Model 1: edge(X1, X2). edge(X3, X4) ...
   Model 2: edge(Y1, Y2). edge(Y3, Y4) ...

   =>Create nodes:  X1, X2, X3, ..., Y1, Y2, Y3, ...
   =>Create edges:  Create a line between the first term and
   the second term.
    X1 -> X2, ...
   ```

   Duplicated nodes or edges are used only once.

2. Generate n graphs: The algorithm to create n graphs is the same as shown in the first point. The difference is to apply this algorithm to each answer set seperately.

This algorithm is applied to edges as well as highlight-edges. Answer sets (nodes) are even charted if there are no relations from or to this node. A highlight-edge must even be drawn if there is no edge between the two nodes from the highlight-edge.

In Figure 3.4 the result graph of the running example is shown. The thin line is the visualization of the edge(X, Y)-predicate from the visualization-output. In contrast the thick line stands for the edgeVis(X, Y)-predicate.

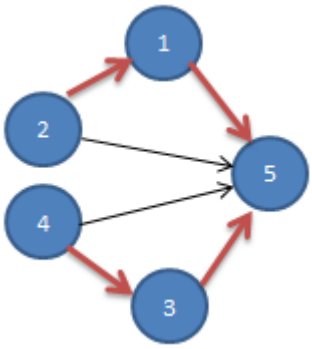

Figure 3.4: Example - Generated result graph

CHAPTER 4

The Answer Set Relationship Visualizer (ARVis)

4.1 Introduction

ARVis is a software tool that represents relations between answer sets as a graph. The previous chapter illustrated the idea of the mechanism (four steps) of visualizing the relations between answer sets. We now present the tool itself. Figure 4.1 shows the main view after executing ARVis. The view consists of a template that is divided into three areas:

1. NORTH: The fist area contains a classical start-menu and a bar that represents all steps to go through with the wizard.
   - Menu: The start-menu contains two options. Both commands can be called from each wizard-step. The 'Reset'-Option is used to clear all input made in ARVis. After selecting this command each input in the program is emptied and the wizard starts from the first step. The configuration-command is described in detail in Section 4.2. Configuration allows specifying the as-solver. Changing of configuration settings takes influence on the program immediately after stepping forward or backward to a wizard page.
   - Wizard-Steps: This bar is an extra feature displaying the current step by highlighting the step name with its border. To prevent inconsistent program-states it is not possible to switch between more than one step forward or back (steps are not click- or selectable).
2. CENTER: The center object is individually adapted to the current wizard-step. Figure 4.1 shows the first step (Execution Base-Program): It allows the selection of files to be solved with an as-solver.

3. SOUTH: In the south of the view a wizard-navigation-bar with navigation buttons is embedded. We can go one step forward with the 'Next'-Button and one step back with the 'Back'-Button. The 'Cancel'-Button exits ARVis.

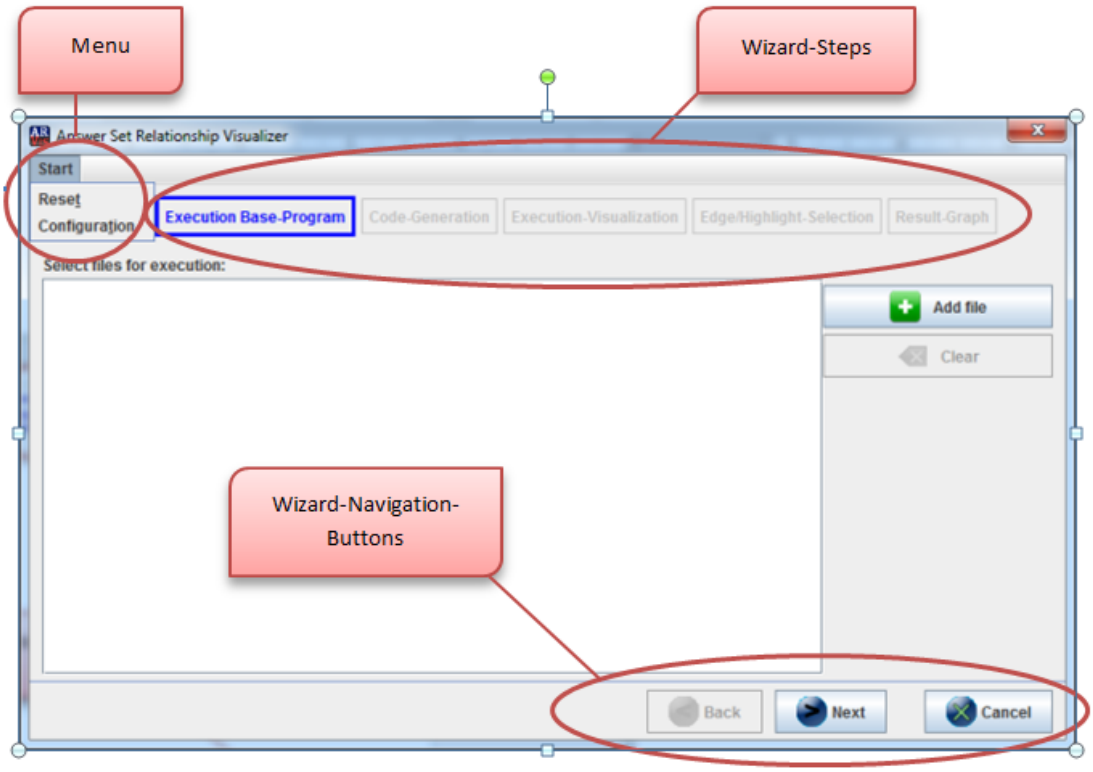

Figure 4.1: ASP Visualizer - Main View

The following example stretches through this whole chapter to describe the functionalities of ARVis. It is based on dlv-syntax. In order to illustrate the capabilities of ARVis we adapted the original problem of the traveling salesman problem. First we describe the traveling salesman problem and in the next step the adaptations.

Example. The target of the traveling salesman problem is to find the shortest path between an amount of cities. The requirements are:

1. Each city must be visited exactly once.
2. The start-city and the end-city must be the same.

Adaptations for ARVis:

1. Find for each country the city with the most inhabitants in a database. This step is illustrated in Figure 4.2 by generating a node (biggest city) out of a city-list.

2. Generate flights between the biggest cities. This step is also presented in Figure 4.2 by the arrows (edges) between the biggest cities.

3. Compute all paths that are a solution for the traveling salesman problem.

4. Visualize all computed paths and highlight the path with the minimal costs.

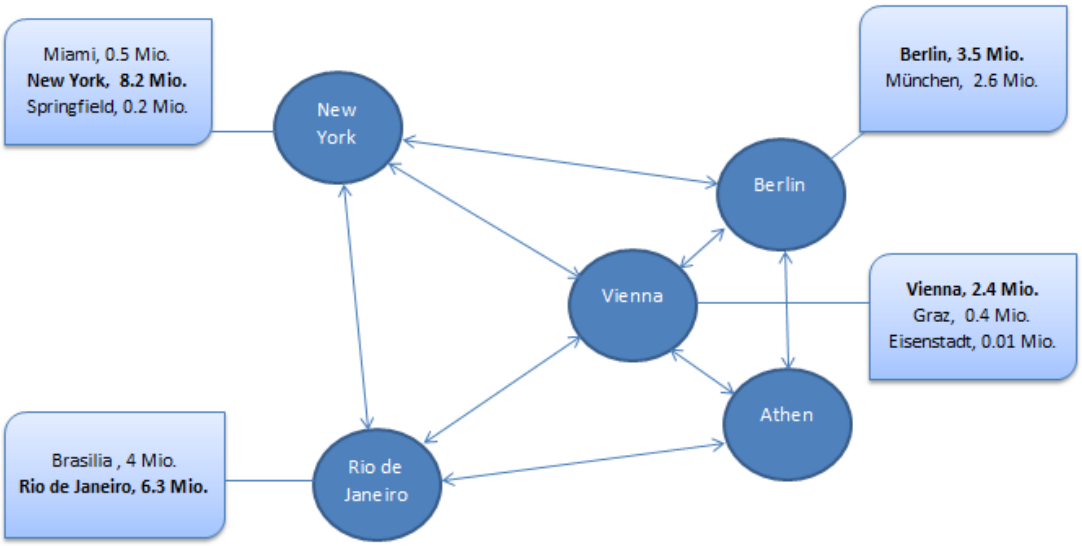

Figure 4.2: Example - Traveling Salesman Problem

4.2 Configuration

General

The configuration is generally used to set up adjustments to ARVis. The first main task of the configuration allows the selection of the as-solver and setting general solver-parameters. The second important task specifies the visualization structure of the result graph (generate only 1 graph). Figure 4.3 illustrates the Configuration-Dialog. The following settings can be changed:

1. Number of models: The upper limit of the models computed by the solver can be set. Complex calculations require a higher amount of models than simple calculations. This number of models can be set up for the first (step 1: for execution) and the second (step 3: for visualization) execution of the solver. This setting has only influence on the dlv-solver.

2. AS Solver: It is possible to choose between different solvers (DLV, Clingo, Others).

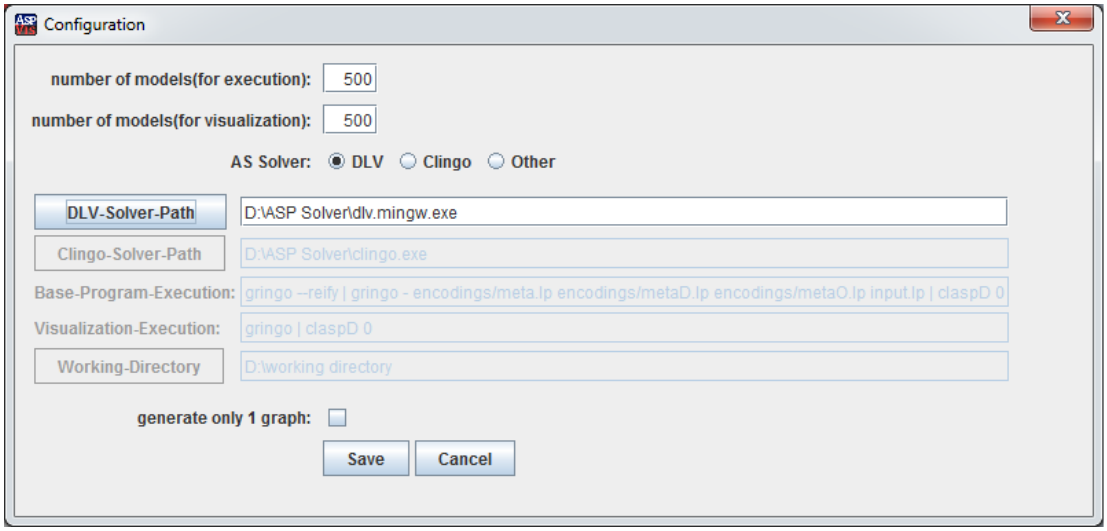

Figure 4.3: Configuration - General Description

3. generate only 1 graph: This setting specifies whether one or more models are visualized at the end of the wizard. In the subsequent example the consequences of this option are illustrated.

DLV

The first choosable solver is the dlv-solver [1]. It arised from the educational area. DLV is based on disjunctive logic programming and offers front-ends for knowledge representation formalism.

It is neccessary to specify the path to the dlv-solver. Therefore a file dialog is implemented which allows for the selection of the dlv execution-file on the hard disk.

Clingo

Clingo provides the advantages of gringo and clasp. [9] provides detailed information about Clingo. The steps from the grounder to the solver are:

1. Execute a program by gringo => Ground
2. Pass the result with a pipeline to clasp.
3. Generate end-result => Solve

These steps are combined by clingo. The selection of the path to the solver is also required (see dlv-solver).

Other

To assemble a custom-build solver an arbitrary amount of grounder- and solver-components can be chosen. A grounder component, like gringo, computes a variable-free program. The output can moreover be applied to an answer set solver like clasp or claspD. It is important to know that the result-output of the grounder must have a format the solver could interpret. Furthermore the output format (end-result) is specified as follows:

```
Answer: 1
predicate1(x) predicate2(y) ... predicateN(z)
Answer: 2
...
Answer: n
...
SATISFIABLE | UNSATISFIABLE | UNKNOWN
```

In order to obtain the result, the parser of ARVis begins with the first 'Answer' and stops as soon as 'SATISFIABLE', 'UNSATISFIABLE', 'UNKNOWN' or the end of the output is reached.

There are three settings that can be manipulated (shown in Figure 4.4):

1. Base-Program-Execution: An user-defined amount of programs can be executed together. The syntax of the input can be adhered to the shell-command-syntax. The possibiltiy of executing an arbitrary amount of programs is given by piping the arguments. The output from one program is streamed as input for the next program. Note that any input files that are specified by the user in the first wizard step are appended directly to the first command (e.g.: file paths are appended after gringo –reify).

2. Visualization-Execution: The syntax and format of this option is equal to the first option (Base-Program-Execution). The sole difference is that this execution is used for the third step of generating a graph to show the relations between answer sets.

3. Working-Directory: This directory specifies the location of files and executables used within the command inputs. It is used as base of all commands that start with clingo, gringo, clasp or claspD. Furthermore it defines the directory of each occurence of a path or file in the arguments of the command. The working-directory has no influence on absolute defined paths in the commands.

4.3 Execution Base Program

In Figure 4.5 the main mask is shown if you start ARVis. The first step is to select input-files to be solved by a solver chosen in the configuration.

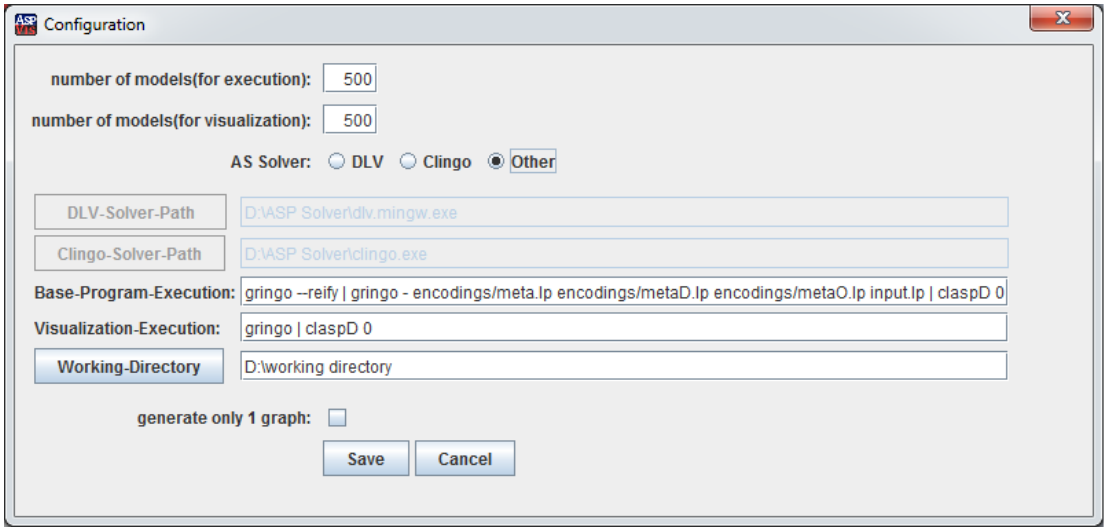

Figure 4.4: Configuration - Other

With 'Add file' a file-selection-dialog appears. It supports the selection of several files that should be appended to the file list. The 'Clear'-Button empties the file list. After execution with the 'Next'-Button an error could appear that shows mistakes made in an as-code-file or in the configuration. The wizard abides by the first step.

Example. For our running example we provide the following two files. The first input-file (input.dl, also shown in Figure 4.5) contains the information about the cities (inhabitants, country of the city).

```
1  Example - input.dl:
2
3  % city(city_name, inhabitants). inhabitants...unit = 1:100000
4  city(ny, 82). city(springfield, 02). city(miami, 05).
5  city(vienna, 24). city(graz, 04). city(eisenstadt, 01).
6  city(rio, 63). city(brasilia,4). city(berlin, 35). city(muenchen,26).
7  city(athen, 40).
8
9  % city_country(city_name, country_name).
10 city_country(ny, usa). city_country(springfield, usa).
11 city_country(miami, usa). city_country(rio, brasilien).
12 city_country(brasilia, brasilien). city_country(vienna, austria).
13 city_country(graz, austria). city_country(eisenstadt, austria).
14 city_country(berlin, deutschland). city_country(muenchen, deutschland).
15 city_country(athen, griechenland).
```

The second file (biggest_city.dl, also shown in Figure 4.5) calculates the biggest city of each country.

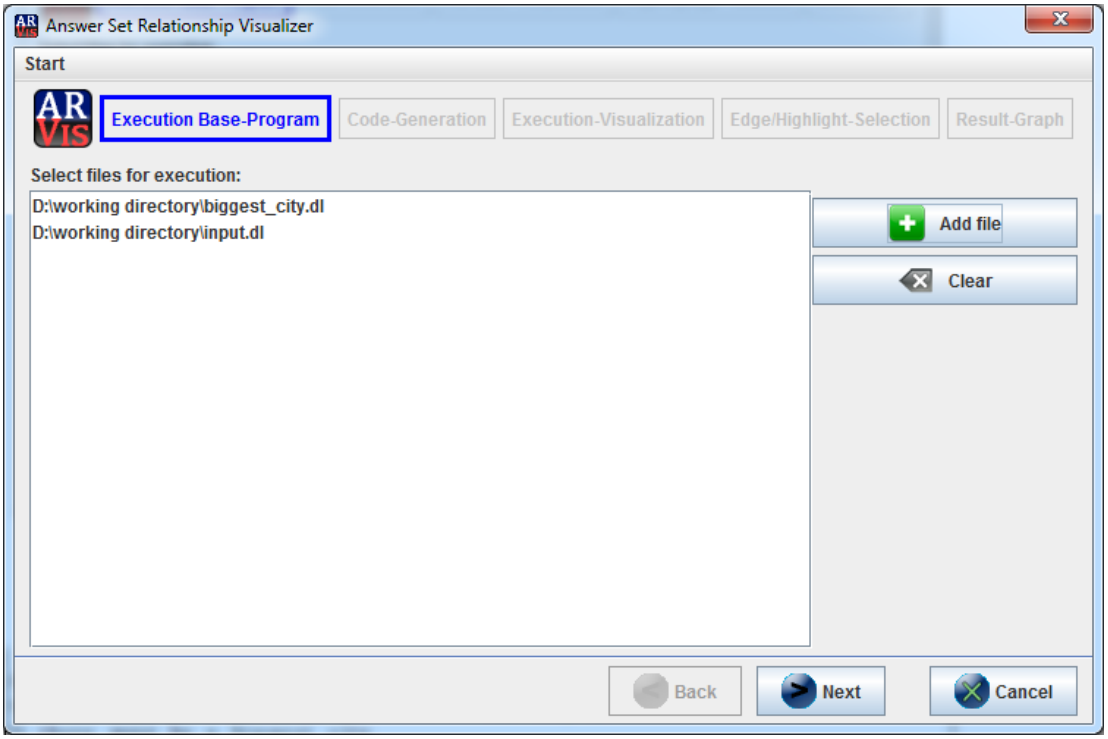

Figure 4.5: Execution Base-Program - Mask

```
1  Example - biggest_city.dl:
2
3  % compute the biggest city from each country
4  biggest_city(X) v -biggest_city(X) :- city(X, _).
5
6  % No population of another city Y from the same country is allowed
7  % to be higher thant the population of the city X.
8  % there must be a biggest city
9  :- biggest_city(X), city(X, E1), city_country(X, C), city(Y, E2),
10    city_country(Y, C), E2 > E1.
11 :- biggest_city(X), biggest_city(Y), X <> Y.
12 :- #count{X : biggest_city(X), city(X, _)} = 0.
```

4.4 Code Generation

After the as-solver execution the computed models are shown in an output-text area. This text area can not be modified (should prevent errors). Now the predicates from the result-models should be selected (multiselection) that are pulled up for the ASP-code generation. By additionally pressing 'Strg' while selecting predicates muliple rows can be selected. In order to deselect

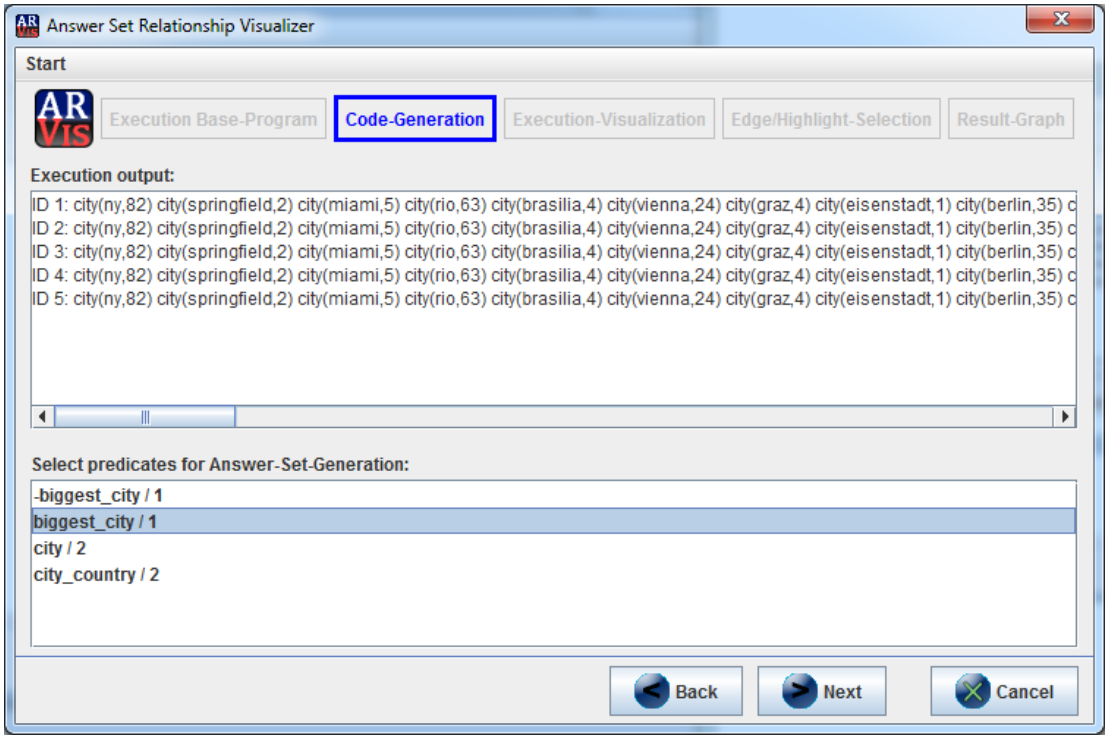

Figure 4.6: Code-Generation - Mask

an item press 'Strg' and click it. This line of action can be applied to any list in the whole ARVis-program. With a click on the 'Next'-Button the code-generation starts.

4.5 Execution Visualization

This wizard step shows the generated ASP-Code that serves as an input for the visualization. These generated literals build the input-instance for the further computations. The first parameter of the literal 'as' identifies an answer set generated in the previous step. This identifier will be used for the graph-visualization as representative of the node by this id.

As described in the first step we have to choose visualization-files that produce the relationships between the generated answer sets. The approach for file-selection and error-handling is the same as described above.

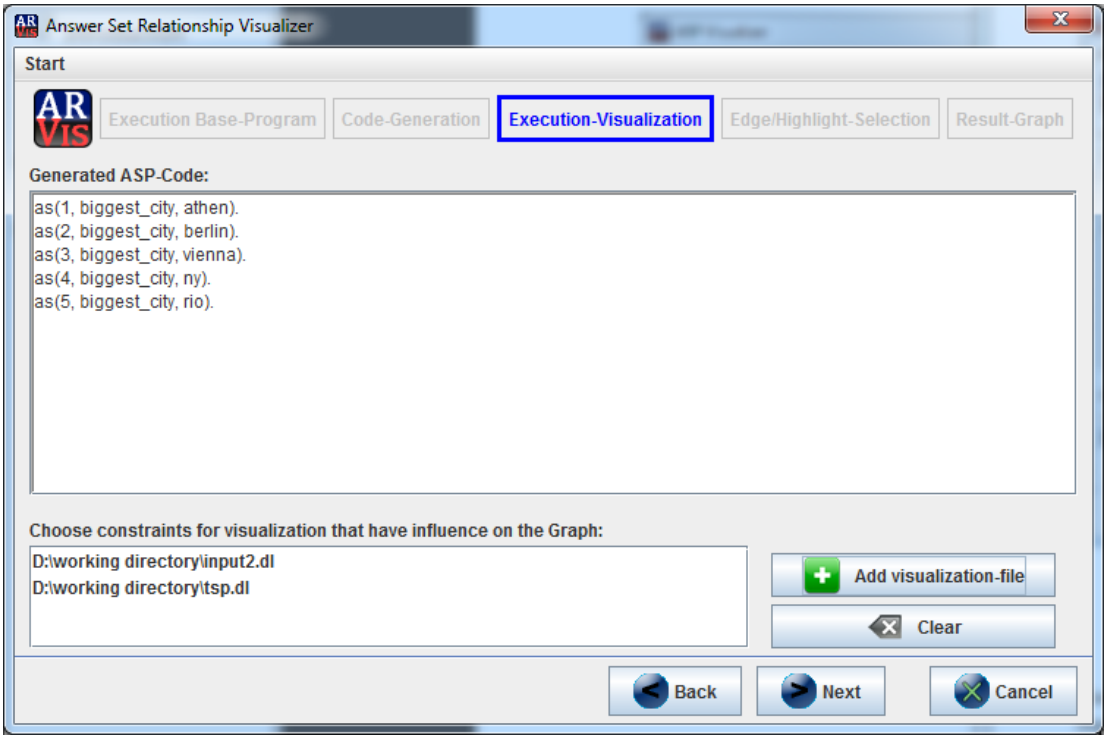

Figure 4.7: Execution-Visualization - Mask

Example. In the files below (input2.dl, tsp.dl) the extra input (input2.dl) and the ASP-Program (tsp.dl for solving the traveling salesman problem) are given.

```
Example - input2.dl:

% possible routes with the associated costs
start(5).
flight(ny, rio, 70). flight(rio, ny, 60).
flight(ny, berlin, 80). flight(berlin, ny, 70).
flight(rio, vienna, 40). flight(vienna, rio, 50).
flight(rio, athen, 60). flight(athen, rio, 70).
flight(vienna, berlin, 10). flight(berlin, vienna, 20).
flight(berlin, athen, 30). flight(athen, berlin, 40).
flight(vienna, athen, 50). flight(athen, vienna, 40).
flight(vienna, ny, 30).
```

```
Example - tsp.dl:

% mapping from the city to the answer set id
as_flight(A, B, C) :- flight(X, Y, C), as(A, biggest_city, X), as(B,
    biggest_city, Y).

% guess a path that starts at start(X) and contains each node once
path(X, Y, C) v -path(X, Y, C) :- start(X), as_flight(X, Y, C).
path(X, Y, C) v -path(X, Y, C) :- reached(X), as_flight(X, Y, C).

% Remember that a node is reached
reached(X) :- path(Y, X, _).

% Check whether there are not two paths from a node
:- path(X, Y1, _), path(X, Y2, _), Y1 <> Y2.
% Check whether there are not two paths to a node
:- path(X1, Y, _), path(X2, Y, _), X1 <> X2.
% Check whether each biggest city of a country is reached
:- as(X, biggest_city, _), not reached(X).

% Mapping from the path to the edges of the graph
% This is done because the predicate must have arity 2
edge(X, Y) :- path(X, Y, _).


% Compute the best path for the traveling salesman person
% Evaluation is the same
best_path(X, Y, C) v -best_path(X, Y, C) :- start(X), as_flight(X, Y, C).
best_path(X, Y, C) v -best_path(X, Y, C) :- reached2(X), as_flight(X, Y, C).
reached2(X) :- best_path(Y, X, _).
:- best_path(X, Y1, _), best_path(X, Y2, _), Y1 <> Y2.
:- best_path(X1, Y, _), best_path(X2, Y, _), X1 <> X2.
:- as(X, biggest_city, _), not reached2(X).
% Calculation of the best path by the use of weak constraints
:~ best_path(X,Y,C). [C:1]
% Mapping from the best_path to the edgeVis-predicate
edgeVis(X, Y) :-best_path(X, Y, _).
```

4.6 Edge/Highlight Selection

The text area (not mutable) in Figure 4.8 represents the answer set output of the solver. The last step before graph visualization includes the selection of two predicates with an arity of 2. The edge-predicates define the connection between the nodes. Highlight-predicates define the same with the difference that they are used to emphasize a special relationship between two nodes. It is not required to select edge- or highlight-predicates. As a result no connections would be drawn if no selections are made.

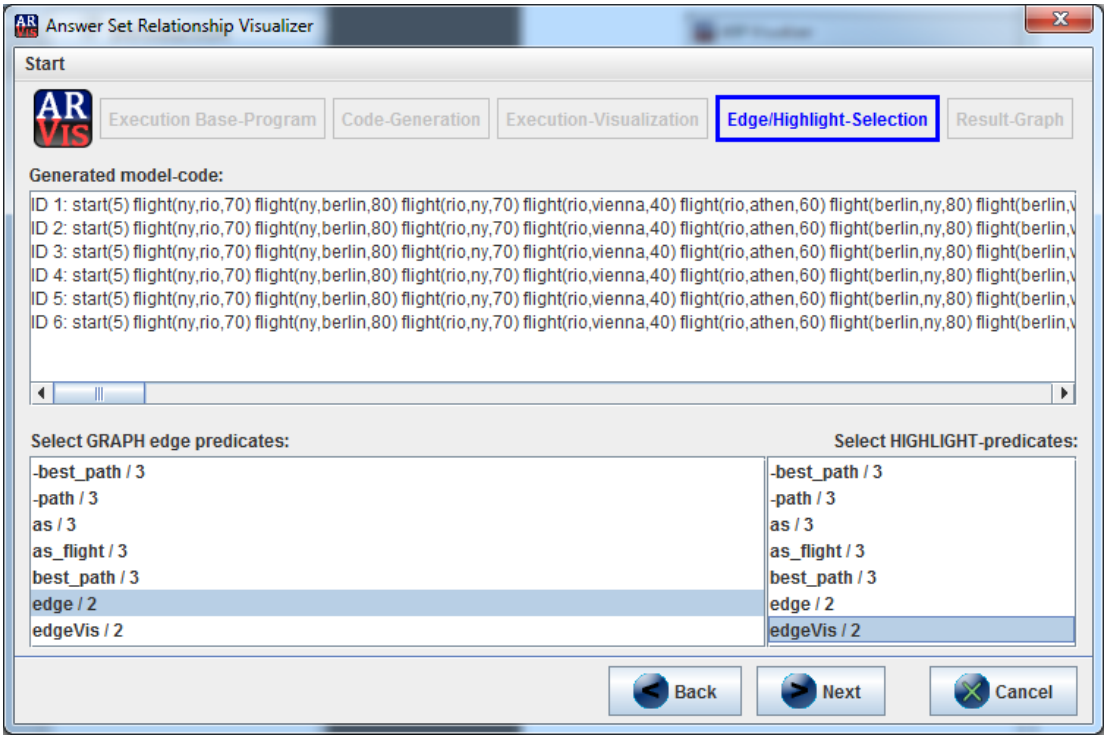

Figure 4.8: Edge/Highlight-Selection - Mask

4.7 Result Graph

Overview - Several Graphs

The final step illustrates the end result of the tool that visualizes the relations between answer sets in a graphical way. Depending on the configuration of the 'generate only 1 graph'-setting a different computation of the graph is used. In Figure 4.9 one can see the resulting graphs when this option is disabled in the configuration. For each answer set of the 'Execution-Visualization'-step a seperate graph (model) is visualized.

The main view contains 3 areas:

1. Select a model: If the 'generate only 1 graph'-option is disabled more than one model could be the result. The list is used to switch between the calculated models. Depending on the selected model in the list the associated graph is displayed ad-hoc in the Graph-Visualization-Area.

2. Graph-Visualization: Nodes describe the answet sets by their generated id and the edges, illustrated by arrows, show the relations between them. This area is a graph-view that

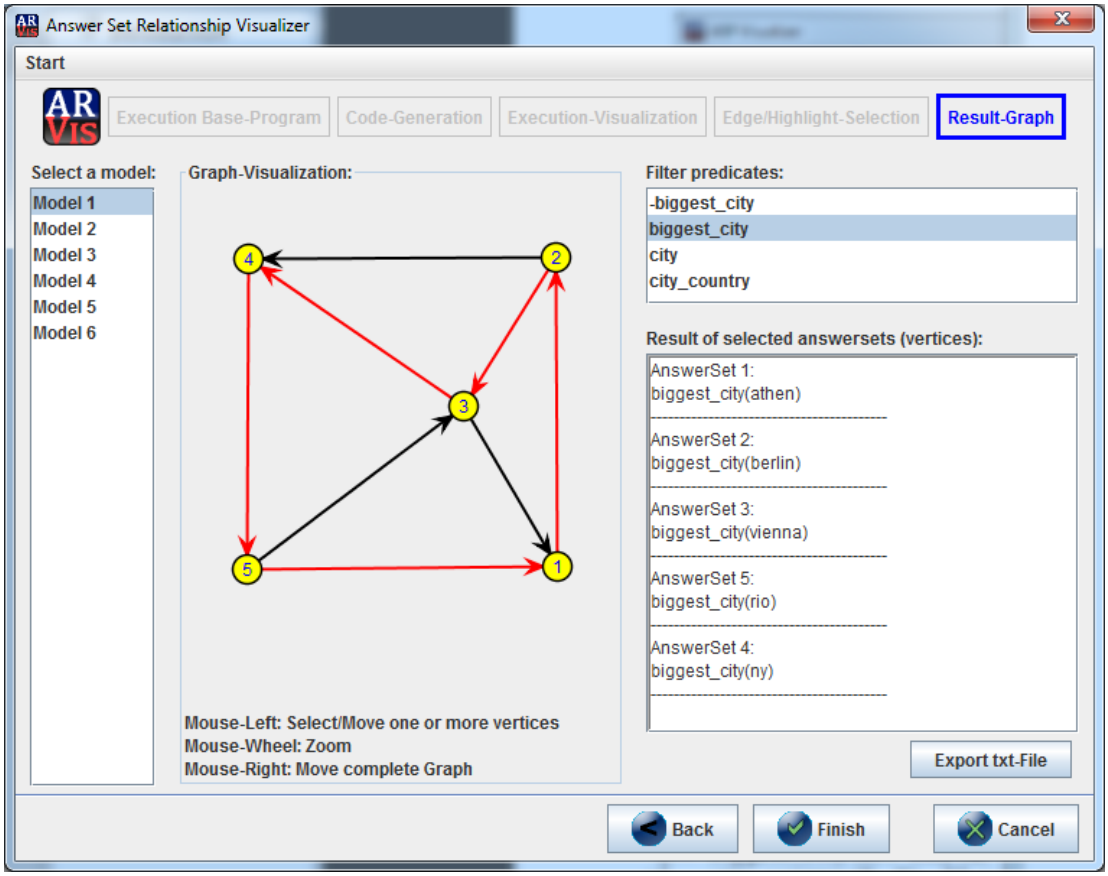

Figure 4.9: Resulting graphs

provides a lot of opportunities. Generally the handling of this view occurs with the mouse. Possibilities:

- Zoom - Zoom from or into the view by scrolling the mouse wheel.
- Move all nodes - Hold the right mouse button and move the whole graph to another position.
- Move selected nodes - Select the desired nodes. After the selection of the last edge hold the left mouse button and move only the wanted nodes. The arrows between the nodes scale automatically.

3. Info-area: Contains the Answer-set-area (for detailed informations) and a Filter-predicates-area. The facilities are explained in subsection Node-Selection and subsection Filter Predicates.

Figure 4.10 shows all computed models from the example in one representation. As one

can see, the third graph that consists of highlighted (red) arrows defines the best path for the traveling salesman problem.

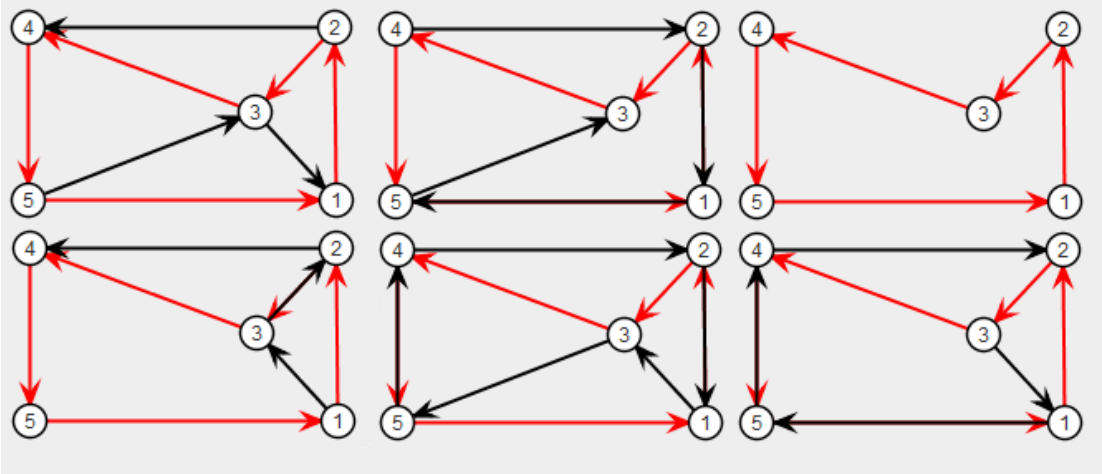

Figure 4.10: Example - Generated Models

Overview - One Graph

Figure 4.11 represents the Result-Graph after enabling the 'generate only 1 graph'-option in the configuration. Provided that at least one model is computed by the solver, only one graph is visualized (Model 1 in the list). Each normal- and highlight-arrow of each model is only drawn once in the graph.

Node-Selection

The generated answer sets, identified by the numeric id, can be detailed in the area 'Result of selected answer sets (=>vertices)'. After (de)selecting one ore more nodes in the view the literals of the answer sets are dynamically shown in the text area. This output area is not mutable. In Figure 4.11 every node is selected and no predicate-filter is set. Figure 4.12 shows that the selection of two nodes entails a specification of the selected nodes in this area.

Filter Predicates

To reduce the details of the answer sets in the text area only to the necessary information, the predicate-filter can be used. An arbitrary amount of predicates can be selected that are filtered in the Answer-set-area. The predicate-filter-list makes the filtering of one or more predicates in each answer set possible. Figure 4.13 represents the filtering of the predicate biggest_city in the text area.

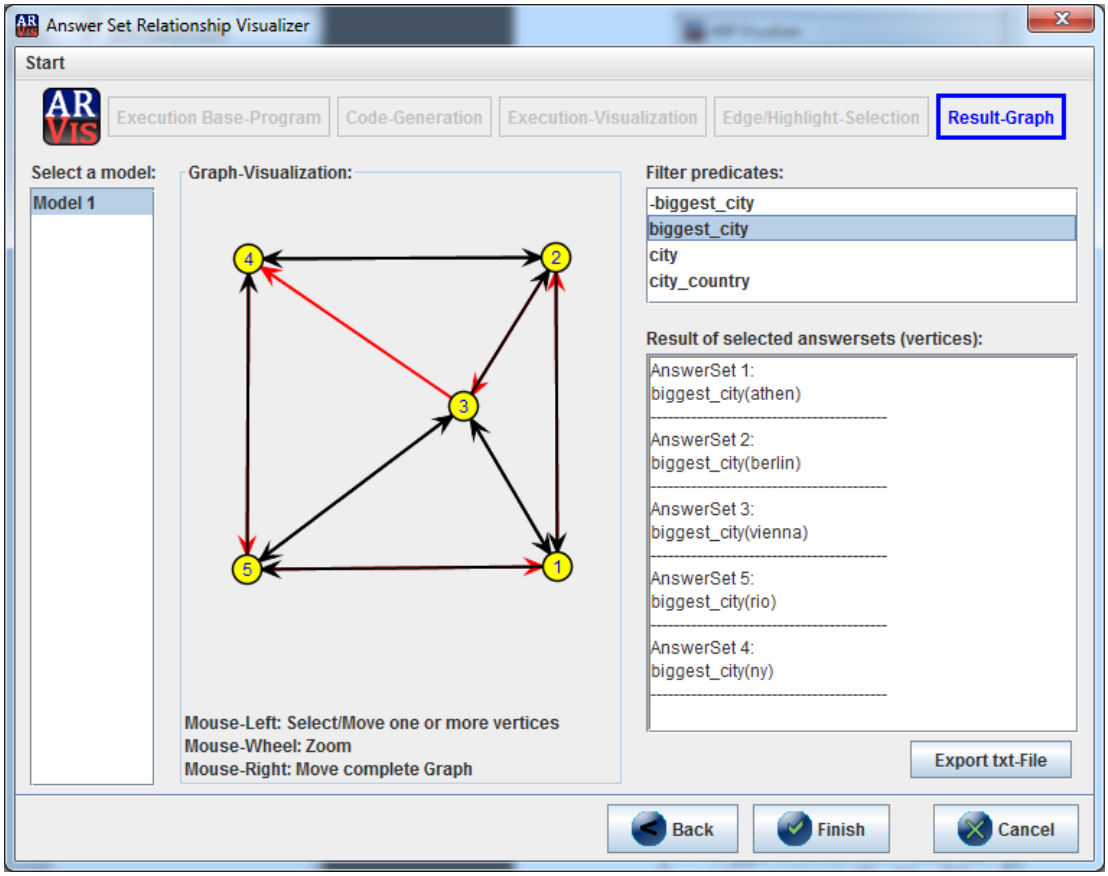

Figure 4.11: Generate only one graph

Export txt-File

To make a backup of the execution result one has to click on the button 'Export txt-File'. This option saves a txt-file with the following information:

- Result after execution with the first solver (first step)
- Result after visualization-execution with the second solver (third step)

After the click on the button the file-save-dialog appears to select a location to save the backup-file. After selecting a location on a storage medium the file is produced and can be shown or manipulated with an external editor-tool (represented in Figure 4.14).

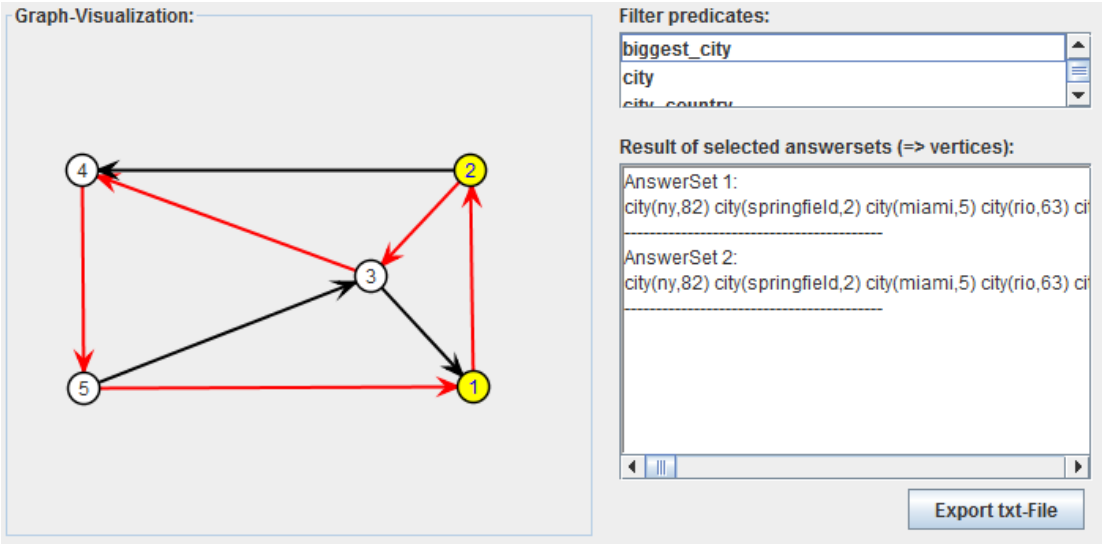

Figure 4.12: Selecting nodes

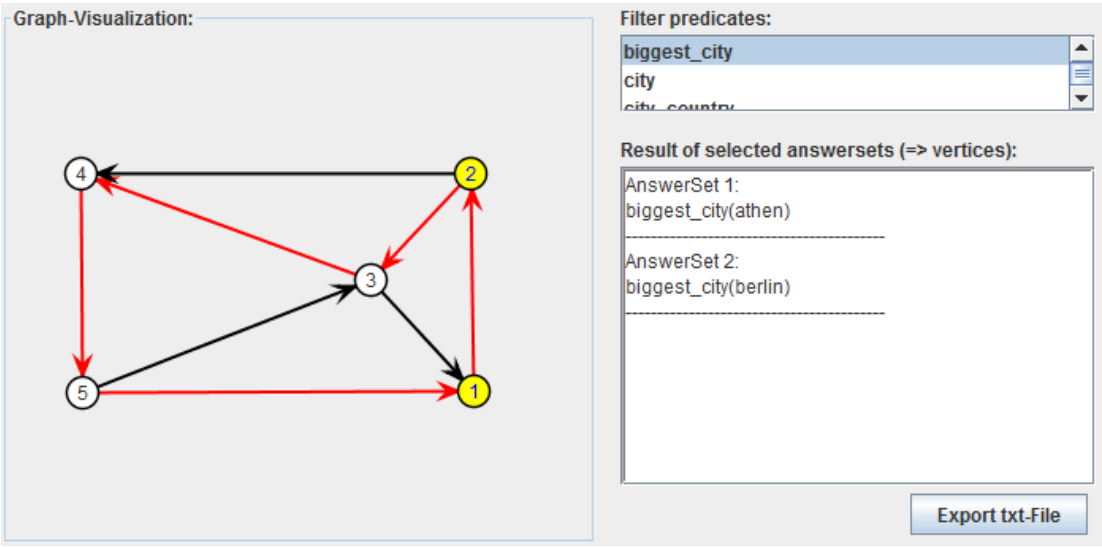

Figure 4.13: Filter predicates

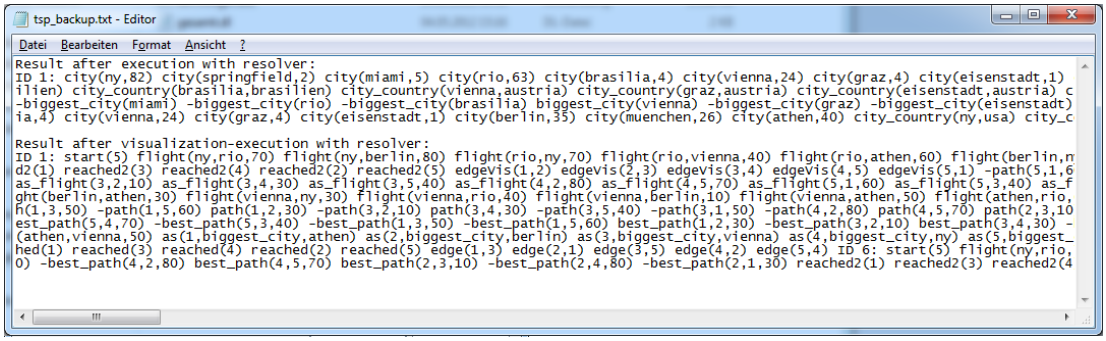

Figure 4.14: Created export-file

CHAPTER 5

Conclusion

In this bachelor-thesis, we illustrated the tool ARVis. On the one hand we described the theoretical mechanism of how to visualize relations between answer sets, and on the other hand we presented a user guide for the tool ARVis. ARVis is a Java standalone-tool that consists of five wizard steps that have to be passed through to get a representation of relations between answer sets as a graph.

Existing visualization tools like ASPViz, IDPDraw, Kara or ASPIDE serve the purpose of better understanding the output (answer sets). Furthermore tools like ASPIDE or Kara support the construction of as-programs with visual editors. Visualization of the relations between answer sets is a completely new mechanism that is founded in this bachelor thesis.

In future work, we intend to add further features to ARVis. The following enumeration lists ideas how the tool can be extended:

- Save current state: We want to give the possibility to save the current program state. Input files, predicate selections and the configuration-settings should be saveable. This has the extra advantage that inputs, selections and settings can be restored after a restart of the program.
- Graphic-Modification: This bachelor-thesis describes the 'forward'-mechanism to get from asp-programs to a 'static' visualization-graph. The other direction to get from a computed graph, that is arbitrarily edited, to input-instances and programs would be a special challange for future works.
- Graphic-Options: A further extension of ARVis is to make the end-graph adjustable. That means that the user has the options to set the node/edge color, node/edge form, node/edge thickness, ...

- Support for further as-solvers: Currently the output formats of DLV and the Clingo/clasp-family are supported. In future work additional output wrappers for further as-solvers should be implemented.
- Graph Export: The current version of ARVis does not support the export of the graph as image. To show or manipulate the graph in an external tool an graph-export interface has to be constructed. Formats like PNG and PDF should be considered for the export.
- Export formated text into file: The current function 'Export txt-File' should be extended by manually configuring output formats. Setting a dot after a predicate would be an example for a format setting.

Bibliography

[1] DLVSYSTEM. http://www.dlvsystem.com/dlvsystem/html/DLV_User_Manual.html, Accessed: 2012-06-12.

[2] Examples for ASPIDE. https://www.mat.unical.it/ricca/aspide/documentation.html, Accessed: 2012-06-27.

[3] Owen Cliffe, Marina Vos, Martin Brain, and Julian Padget. ASPVIZ: Declarative Visualisation and Animation Using Answer Set Programming. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 724–728. Springer, 2008.

[4] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321 – 357, 1995.

[5] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. Aspide: Integrated development environment for answer set programming. In James Delgrande and Wolfgang Faber, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 6645 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2011.

[6] Christian Kloimüllner, Johannes Oetsch, Joerg Puehrer, and Hans Tompits. Kara: A system for visualising and visual editing of interpretations for answer-set programs. In *19th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2011) and 25th Workshop on Logic Programming (WLP 2011)*, pages 152–164, 1843-11-06, 2011. INFSYS Research Report.

[7] Katholische Universität Leuven. Examples for IDPDraw. http://dtai.cs.kuleuven.be/krr/software/visualisation, Accessed: 2012-06-27.

[8] University of Bath. Examples for ASPViz. http://www.cs.bath.ac.uk/ occ/aspviz/, Accessed: 2012-06-27.

[9] Universität Potsdam. The Potsdam Answer Set Solving Collection. http://potassco.sourceforge.net/, Accessed: 2012-06-12.