Sahil Gupta
Computer Science

I. Title:

An Extensible Possible Worlds Explorer for Answer Set Programming

II. Abstract:

The goal of this project was to develop a stand-alone tool to help analyze the solutions (Possible Worlds) of a problem generated using an Answer Set Programming (ASP) reasoner such as Clingo or DLV. In PW-Explorer, we have developed a general and extensible tool which provides this functionality by using an ASP output parser, relational databases, and queries on these databases.

III. Introduction:

The objectives of this research project were as follows:

- To analyze the solutions (Possible Worlds) produced by Answer Set Programming (ASP) reasoners such as Clingo and DLV.
- To develop ways to query, visualize, and interpret this data using relational databases such as SQLite Databases and Panda DataFrames.
- To cluster similar and equivalent PWs to aid in finding fundamental solutions
- To identify and analyze the 'simplest' solutions to an underspecified problem, based on a user-defined distance function between PWs. (Occam's Razor: among competing explanations, the simplest one should be selected)
- To conduct a case study, demonstrating the feasibility of the approach.

We have achieved all of the above objectives by developing an extensible possible worlds explorer for answer set programming. We applied our tool to an automobile taxonomy alignment problem, derived from [1].

IV. Methodology:

The toolset was developed primarily using the Python2 programming language in conjunction with other libraries such as Numpy, Pandas, Antlr, SQLite, Scipy, Matplotlib, Sklearn, Mpld3, Plotly and Networkx. The parser for the Clingo output was generated using a popular parser generator called Antlr4. We extended and adapted the generated parser to be able to implement the functionality critical to our project which was to correctly and efficiently load this information into a relational database such as a Pandas DataFrame.

The SQLite and Pandas queries have been implemented as their own distinct functions which take as arguments the possible worlds to consider, the relations to use and other relevant arguments such as instance definitions, columns/attributes to consider, etc.

A user-definable *distance function* takes as arguments the two possible worlds W_1 and W_2 and then computes a distance d(W_1, W_2) between the worlds, e.g., based on the size of their symmetric difference (worlds are represented as sets of logic atoms, called *facts*). Here as well, the relations to be considered can be specified and redundant attributes in these relations can be disregarded. Users can also define their own distance functions and use a different distance metric. For instance, we implemented the absolute difference in number of *overlaps* as a distance metric for the automobile taxonomy alignment example. These distances are then normalized to a number between 0 and 1 for uniformity across several problems.

For complexity analysis, we used the size of the possible world, i.e., the number of relation instances, as the indicator of the complexity of a world. The complexity analysis metric can also be user-defined. For instance, we defined complexity of a possible world as the number of overlaps in the possible world in the taxonomy alignment example. The complexity measurements are normalized to a number between 0 and 1 just like the distance metric above to maintain uniformity across several problems.

The distance function is primarily used to compute a distance matrix (of the possible worlds) which is then subsequently used for clustering these possible worlds using libraries such as scipy.clustering, scipy.linkage, etc. The distance matrix is also used for projecting these possible worlds on a 2D plane by libraries such as Networkx (which uses neato to achieve this).

The complexity definition is primarily used to rank the possible worlds in order of their complexity, which then allows us to identify the simplest and the most complex set of solutions. These complexities can also be used in the distance calculations directly, such as in the taxonomy alignment problem in section V where the absolute difference between the complexities of two possible worlds is used as the distance between the two possible worlds.

For generating the visualizations, we used tools such as Matplotlib, Scipy, Plotly and Networkx. The former three were used to identify clusters of similar possible worlds and visualize them in the form of Dendrograms. Networkx was used to map the possible worlds on a 2D plane using the distances between each pair of possible worlds.

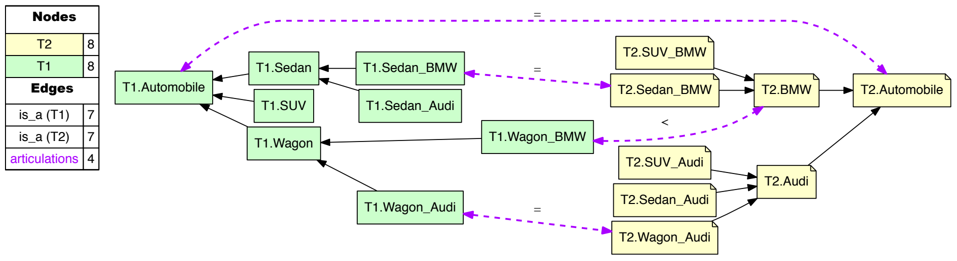
We applied our approach to an automobile taxonomy alignment problem described in [1] to test its functionality.

V. Automobile Taxonomy Alignment Example

This example is explored by Salvatore Raunich and Erhard Rahm in their research paper titled ‘Target-driven merging of taxonomies with ATOM’ [1].

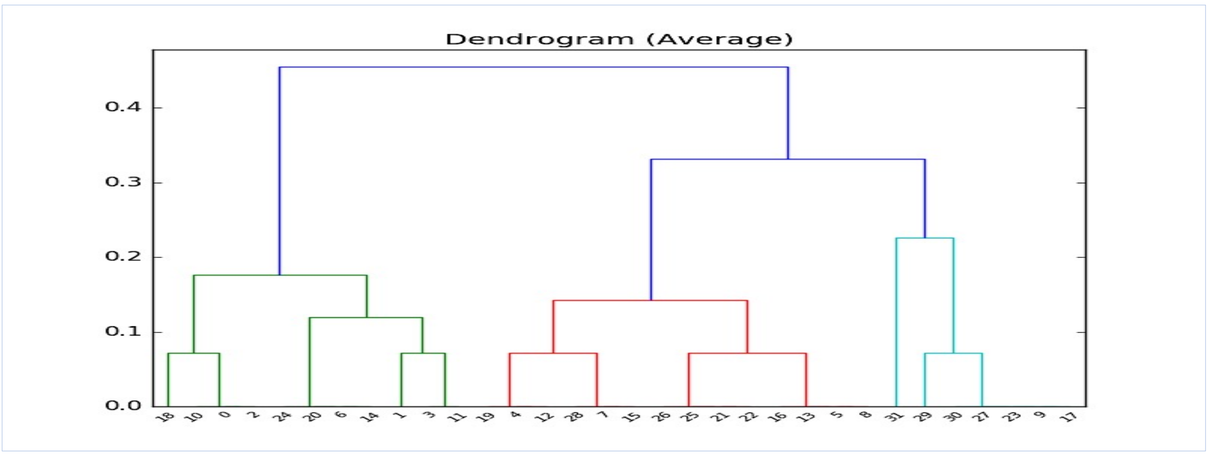
The following is a graphical visualization of the provided input articulations:

Sahil Gupta
Computer Science



There are 32 solutions/possible worlds to this problem.

The PW-Explorer organized them in 8 different groups. The Dendrogram depicting the hierarchical division looks as follows:



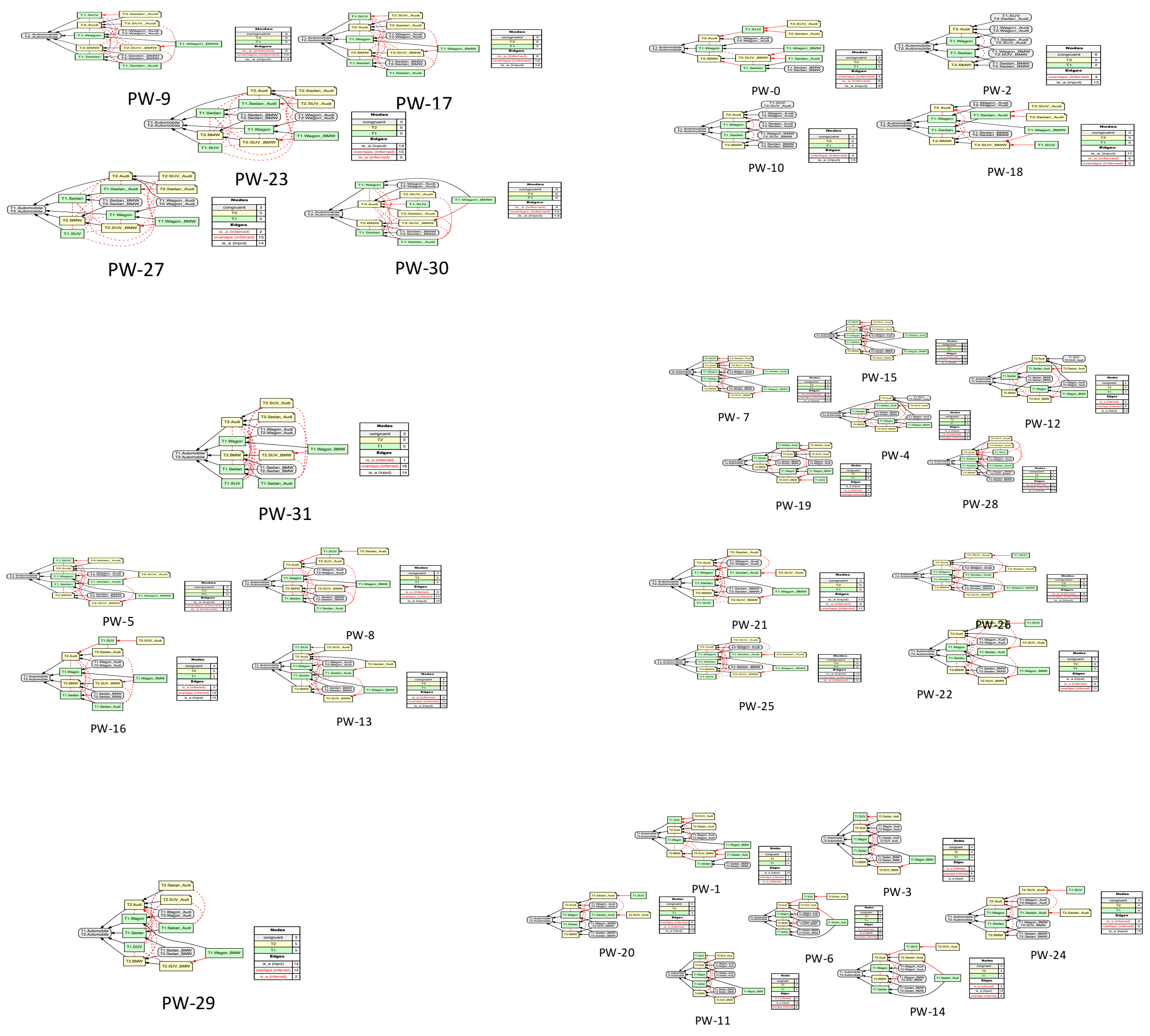
The custom distance metric used in this example is the absolute difference in the number of overlaps in the possible worlds. The custom complexity definition used was the number of overlaps (><) in the possible world. It was computed using the following query:

select count(*) **from** rel_3
where x3 = "><"
and pw = pw_id;

where pw_id ∈ [0,31], one for each of the possible worlds.

The following were the computed complexities in decreasing order:
PWs: [31, 9, 17, 23, 27, 30, 29, 5, 8, 13, 16, 21, 22, 25, 26, 7, 15, 28, 4, 12, 19, 3, 11, 1, 6, 14, 20, 24, 18, 0, 2,10]
Complexities: [1.0, 0.79, 0.79, 0.79, 0.79, 0.79, 0.71, 0.57, 0.57, 0.57, 0.57, 0.5, 0.5, 0.5, 0.5, 0.43, 0.43, 0.43, 0.36, 0.36, 0.36, 0.29, 0.29, 0.21, 0.14, 0.14, 0.14, 0.14, 0.07, 0, 0, 0]

The following 8 clusters were identified by the tool based on our custom distance definition:



As can be observed, the possible worlds in each cluster have similar complexity in terms of the number of overlaps in these possible worlds. They are also very similar in terms of the relations inferred within them.

VI. Results

With the PW-Explorer tool we have implemented the functionality to:

1. Parse the output of Clingo, a widely-used ASP reasoner
2. Populate it in relational databases such as SQLite Databases and Pandas DataFrames
3. Run powerful, highly-customizable queries such as intersection, union, frequency of an instance of a relation, number of instances, difference, symmetric difference, redundancy detection and so on.
4. Calculate distances between these Possible Worlds using either basic distance metrics such as symmetric set difference or custom, context-specific and user-

defined distance metrics such as absolute difference in number of overlaps in case of taxonomy alignment.
5. Use these distances to group similar or possibly equivalent possible worlds together to identify fundamental solutions of a problem
6. Analyze the complexity of these solutions using either basic criterions such as size of the Possible World or custom criterions such as number of overlaps in case of taxonomy alignment.
7. Identify the 'simplest' solutions (cf. Occam's Razor: among competing solutions, choose the simplest ones).
8. Better interpret the relationship between these Possible worlds using visualizations such as Dendrograms, 2-D Projection of the Possible Worlds, etc.

We applied this approach to the autonomy taxonomy alignment problem and found that similar solutions were in fact grouped together as expected. We were also able to run powerful queries across these possible worlds.

VII. Discussion

As the taxonomy alignment example above demonstrates, the PW-Explorer can be used to analyze various answer set problems. We have used the tool for other problems such as job assignment, n-queens, graph coloring and so on. These are all problems with multiple possible solutions and therefore, fit to be analyzed by a tool such as ours.

The PW-Explorer can be easily extended with custom and context specific visualization techniques, distance metrics and complexity definitions. For instance, the EulerX tool used for taxonomy alignment could use PW-Explorer for analyzing alignments by producing the required Clingo problems, defining appropriate distance and complexity notions and using tools like graphviz to visualize the possible worlds appropriately. To draw a parallel from Object-oriented programming, the PW-Explorer is like a base class that can be extended to create powerful situation specific children classes.

VIII. Conclusion

PW-Explorer has been implemented as an open source tool on GitHub; all the software is available here [5]. PW-Explorer provides a general stand-alone tool that helps analyze solutions produced by Answer Set Programming (ASP) reasoner Clingo. It can be used to analyze these possible worlds effectively and efficiently using powerful SQLite and Python/Pandas queries. It can also be used to group similar solutions together and to also identify the simplest solutions in underspecified problems.

IX. References

1. Raunich, Salvatore, and Erhard Rahm. Target-driven merging of taxonomies with ATOM. *Information Systems* 42 (2014): 1-14.
2. Thau, David, Shawn Bowers, and Bertram Ludäscher. Merging Taxonomies under RCC-5 Algebraic Articulations. *Journal of Computing Science and Engineering* 3, no. 2 (2009): 109–26.
3. Franz, Nico M., Mingmin Chen, Parisa Kianmajd, Shizhuo Yu, Shawn Bowers, Alan S. Weakley, and Bertram Ludäscher. Names are not good enough: Reasoning over taxonomic change in the Andropogon complex. *Semantic Web Journal* 7, no. 6 (2016): 645-667.
4. EulerX Tool: https://github.com/EulerProject/EulerX
5. PW-Explorer Tool: https://github.com/idaks/PW-explorer