The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM)
ISSN: 2602-3199

The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM), 2021

Volume 12, Pages 85-94

ICRETS 2021: International Conference on Research in Engineering, Technology and Science

IMSD: Interactive Methods for Finding Similar or Diverse Answer Sets

Asmaa AFEEFI
An-Najah National University

Abstract: Answer set programming (ASP) is a modeling language in knowledge representation, rooted in Logic Programming and Nonmonotonic Reasoning, which has been gaining increasing attention during the last years. In recent years, many of the researchers developed integrated development environments (IDE) for ASP programs including editors and debuggers. Other researchers focused on analyzing the answer sets, they introduced offline and online methods to find specific solutions of a given problem in answer set programming in different approaches such as phylogeny reconstruction. However, with an enormous number of answer sets could be available, the user is not interested in all of them. Thus, a navigation of the search space could be a solution to help the user to access the specific answer sets. To this end, we aim at finding similar/diverse solutions of the answer sets with a new method. The intuition behind this navigation is to make the search faster than other methods and explore information that is related to the user's query. Afterward, we implement a tool performing the above approach in order to simplify the search task and show the applicability and effectiveness of our method. We conclude by testing the performance of the proposed tool into a real-world example of ASP programs.

Keywords: Answer set programming, Navigation approach, Diversity, Similarity

Introduction

It is worth finding solutions of the answer sets which are similar/diverse to each other. For instance, in planning, it might be useful to compute a set of similar plans. Therefore, when the execution of the plan fails, one can switch to a very similar one. Towards this goal, we represent a problem at hand by a logic program, such that its answer sets correspond to solutions. These solutions characterize the solutions of the original problem, and then, use an answer set solver to find such solutions. In the last few years, many solvers are developed, such as, Clasp (in conjunction with Gringo), DLV, Clingo (Gebser et al., 2014), and SMODELS (*Computing the Stable Model Semantics*, n.d.).

On one hand, the researchers have turned their attention to develop different integrated development environment (IDE) for ASP programs including editors and debuggers (e.g., APE (Fandinno et al., 2019), iGROM, SeaLion (Busoniu et al., 2013)), and like the online development environment for answer set programming (Marcopoulos et al., 2017). On the other hand, some of them have developed tools to visualize the answer sets and their relations by means of a directed graph, such as, ARVi tool (Ambroz et al., 2013).

Despite these improvements, there is a lack of attention to analyze the answer sets themselves. In some particular problems, a massive amount of answer sets could be available. However, the user is not interested in all of them. In (Afeefi, 2019) we implemented different navigation approaches, such as, one case of finding diverse/similar solutions to help the user to access the specific answer sets. To this end, we are looking into another two cases for finding diverse/similar solutions. The intuition behind this navigation method is to make the search faster and explore information that is related to the user's query.

Related Work

Analysis of answer-set programming (ASP) is one wide field that is increasingly growing in the last few years. At first, several tools have been designed to support the user in developing ASP applications, and the visualization aspects of these tools focus on the representation of single answer set. Eiter et al introduce offline and online methods to find similar or diverse solutions of a given problem in answer set programming in phylogeny reconstruction (Eiter et al., 2009). They study two kinds of computational problems related to finding similar/diverse solutions of a given problem, in the context of ASP: one problem asks for a set of n solutions that are k- similar (resp. k-diverse), the other one asks for a solution that is k-close (k-distant) to a given set of solutions.

On the other hand, different tools for developing ASP programs have been proposed including editors and debuggers. Koziarkiewics implemented iGROM (*IGROM Download | SourceForge.Net*, n.d.) which is an IDE for ASP programs specifically those written in DLV (and its frontends) and Smodels. It provides some features, such as syntax highlighting for DLV and its dialects, error detection for DLV and dialects.

In (Sureshkumar et al., 2007) Sureshkumar et al. implement an Integrated Development Environment (IDE) for ASP, the AnsProlog* Programming Environment (APE). It offers many features, like syntax highlighting, automatic syntax checking, integration of editor; LPARSE and SMODELs, and display dependency graph of program.

Recently, Oetsch et al. in (Busoniu et al., 2013) design an IDE for ASP (SeaLion) as a plug-in for Eclipse platform. This tool provides source-code editors for the languages of Gringo and DLV. It offers functionalities, like syntax highlighting, syntax checking, code completion, visual program outline, and refactoring functionality.

Ambroz et al. (Ambroz et al., 2013) present a new tool, ARVis. The main purpose of ARVis is to visualize answer sets and their relations by means of a directed graph. The general idea for this tool is passing the answer sets of a first user-specified ASP encoding to a second user-specified encoding which specified the relations between them. Obviously, ARVis is not designed to obtain a high performance since a potential exponential number of answer sets of the first program has to be processed by the second one.

As discussed above, some approaches are much more focused on editing and debugging ASP programs. Others are developed for a certain problem. To the best of our knowledge there does not exist a tool yet that is capable of navigating the space of answer sets for general problems.

Computing Similar/Diverse Solutions

Towards this goal, we study finding similar/diverse solutions in answer set programming. The computation of similar and diverse solutions is symmetric. Thus, we focus on finding the diverse solutions. This section introduces Preliminaries, a preprocessing and so-called (modified) interactive method to compute the diversity of solutions.

Preliminaries

We introduce the graph structures used to internally represent the answer sets. Then, we continue with Hamming and Jaccard distances which are the measures for similarity/diversity of the solutions.

1) Graph: Graphs are common fundamental data structures in knowledge representation. We use graphs to represent a set of objects and the relationship between pairs of objects. A graph is defined as the structure $G = \langle V, E \rangle$ representing a set of vertices V (also called nodes) and a set of edges $E \subseteq V \times V$. There are two types of graphs, directed and undirected. In our work, we consider an undirected graph. All the edges in the undirected graph are bidirectional. A complete graph is a simple undirected graph in which every pair of distinct nodes is connected by a unique edge. The complete graph on n nodes has $n(n-1)/2$ edges.

Given a graph G, each edge in E of G might be associated with a real number, then called its weight. G together with these weights on its edges, is called a weighted graph. We therefore exploit this property to express the weight of the edges by a distance, e.g., Hamming distance or Jaccard distance. In this work, the answer sets are internally represented as the nodes of the graph. The edges are labeled by Hamming or Jaccard distances between pairs of nodes.

2) Clique: A clique in a graph G is a complete subgraph in G, that means, it is a subset S of the vertices V such that each two vertices in S are joined by an edge in G. A maximal clique is a clique with the maximum number of vertices; no more vertices can be added. In this work, we are interested to find a maximal clique to obtain a largest complete subgraph. To demonstrate, if the user needs n answer sets that differ in k atoms, we need to find a maximal clique with size n or greater than n that differ in k atoms. Reporting the maximal cliques of a graph is a major problem arising in graph structures. The output of maximal clique enumeration algorithm may be exponentially sized, so that an algorithm with provably good running time w.r.t. the input size is not possible. However, any algorithm reporting all maximal cliques should be output sensitive.

There is a comprehensive bibliography of clique enumeration algorithms. For instance, Bron-Kerbosch (BK) algorithm (Bron & Kerbosch, 1973) and new algorithms with an alternative strategy based on matrix multiplication (Makino & Uno, 2004). Recently, all papers acknowledged BK algorithm as the best one in practice (Baum, 2004). We choose herein BK algorithm to find diverse/similar answer sets. In (Bron & Kerbosch, 1973), Bron and Kerbosch report two algorithms, version 1 and version 2. Version 2 is an optimization of version 1 based on pivots or fixed points. In this work, we implement Bron-Kerbosch (version 2) algorithm to find maximal cliques of the graph whose nodes are the answer sets.

3) Hamming Distance: Hamming distance is used to measure similarity and diversity between two sequences. It is limited to cases when two sequences have the same length. The Hamming distance is defined to be the number of positions at which the corresponding symbols are different. The sequences may be strings or binary vectors (*Fakecineaste : How to Calculate the Hamming Code*, n.d.). Similarly, for answer sets, the length of two answer sets (number of atoms) should be the same. Each answer set is represented as a vector of boolean values. We compare the first contents of the two indexes in each vector. If they are the same, record a "0", otherwise, record a "1" for that index.

4) Jaccard Distance: A very simple and often effective approach to measure the similarity and dissimilarity between non-empty finite sample sets is the Jaccard index. The Jaccard index (Deng et al., 2012), also known as Jaccard coefficient is used to compare the similarity and diversity of non-empty finite sample sets. The Jaccard coefficient is defined as the size of the intersection divided by the size of the union of the sample sets. The Jaccard distance is complementary to Jaccard coefficient and is obtained by subtracting the Jaccard coefficient from 1. Similarly, we can define the Jaccard distance by dividing the difference of the sizes of the union and the intersection of two sets by the size of the union.

The Jaccard coefficient measures the similarity between the non-empty finite sample sets, but the Jaccard distance measures the dissimilarity between the non-empty finite sample sets. For the answer sets, we use this measure when we have different lengths of the answer sets.

Preprocessing

In the preprocessing method, we compute all the answer sets of an ASP program by running an ASP solver. The answer sets are stored with their cardinality in a database. At the same time, we get and store the set of all ground atoms of the answer sets in a text file. After that, we create a hash mapping data structure which maps each ground atom in the file to an integer number. From the set of all ground atoms, we can check whether is in an answer set or not in order to build the boolean vector for computing distance purpose. We build a complete undirected graph $K = \langle S, E \rangle$ whose nodes S correspond to the answer sets AS and edges $E = \{\{s_i, s_j\} \mid \forall s_i, s_j \in S, s_i \neq s_j\}$ are labeled by a function $L : e \rightarrow \mathbb{N}$ that maps each $e \in E$ to a natural number (the distance), such that, $L(\{s_i, s_j\}) = d(s_i, s_j)$. The distances between the corresponding answer sets are calculated by Jaccard or Hamming distance. Additionally, we store the value of maximum (resp., minimum) distance, denoted by d_{max} (resp., d_{min}) between the answer sets for computing diversity/similarity.

Interactive Method (IM)

We study various problems to find similar/diverse answer sets of the given ASP program. As in illustration, the user can specify the number of the answer sets that differ in a certain number of k atoms. More precisely, if the user needs n different answers and specifies a relational operator (e.g., $\leq$) and k atoms then, the result n should be different in $\leq k$ atoms. There are two distances we use for edges in this work. Hamming distance for the answer sets with the same length, and the Jaccard distance for the answer sets with different lengths. After we build a complete undirected graph K in the preprocessing method, we check whether there exists a complete subgraph (or a clique) of size n in K whose distance is specified by the user. In this work, we find a maximal clique to obtain the largest complete subgraph (Makino & Uno, 2004). In detail, if the user needs n answer sets that differ in k atoms, a maximal clique will be found with size greater than or equal to n. Each node in the maximal clique corresponds to an answer set, so it represents exactly one answer set.

Definition. Let A and B be sets. The set A corresponds to the set B, denoted by $\approx c$, where $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, such that

$$A \approx c\, B \;\; iff \;\; \forall b_i \in B \;\; a_i = b_i$$

where $1 \leq i \leq n$.

We are mainly interested in two cases of problems related to the computation of a diverse/similar answer sets:

Case 1 (n-Most Diverse Answer Sets (resp., n-Most Similar Answer Sets))

Instance. Given a complete graph $K = \langle S, E \rangle$ whose nodes $S \approx c\, AS$ (answer sets) of an ASP program P where $E \subseteq S \times S$, a non-negative integer n, and the value of the maximum distance d_{max}.

Question. Does there exist a set S_1 with the cardinality $|S_1| \leq n$ where a complete subgraph (clique) $K\downarrow_{S_1} := (S_1, E_1 \subseteq S_1 \times S_1)$, $E_1 \subseteq E$ and $S1 \subseteq S$, and the distance of the set S_1, denoted by $d_s(S_1)$, is maximum (resp., minimum) distance between each pair of its elements such that

$$d_s(S_1) = d_{max} \;\; iff \;\; \forall s_i, s_j \in S_1$$

$$d_{max} = max\{d(s_i, s_j) \mid \{s_i, s_j\} \in E\}$$

where $1 \leq i, j \leq n$.

To demonstrate, given a complete undirected graph K whose nodes are the answer sets and the edges are labeled by the distance between pairs of the nodes. The user specifies an integer number n and the value of maximum distance d_{max} (and for minimum distance d_{min}) are stored during the preprocessing method; n is the number of the answer sets that differ in d_{max} atoms. A clique of the size at most n with distance equal to dmax is picked from K. The motivation for finding the clique with distance d is to find corresponding answer sets which are different in d_{max} atoms. Algorithm 1 shows Case 1 of the interactive method.

As an illustration, we find n-most diverse answer sets (resp., n-most similar answer sets) by calculating the maximum (minimum) value of the distance and check whether there exists a maximal clique with this distance. If there is no maximal clique, the distance will be decreased by 1 (resp., increased by 1) until the maximal clique is found (Figure 1).

Notations. We shall denote the graph $K' = \langle S, E' \rangle$. Given a node $u \in S$, $nbrs(u)$ denotes the neighbors of u, i.e. $nbrs(u) = \{v \mid (u, v) \in E'\}$.

Algorithm 1: `IMCase1`

Input: A complete graph $K = \langle S, E \rangle$.
A negative integer n.
The maximum distance d_{max} and the minimum distance d_{min}.
Output: A set S_1 of at most n answer sets whose distance is $d_s(S_1) = d_{max}$.

S_1 = **IMAlgo**($\langle S, E \rangle$, n, d_{max}, $=$)
if $|S_1| = \emptyset$ **then**
  while $|S_1| = \emptyset$ *and* $d_{max} \geq d_{min}$ **do**
    $d_{max} \leftarrow d_{max} - 1$
    S_1 = **IMAlgo**($\langle S, E \rangle$, n, d_{max}, $=$)
return S_1

Algorithm 2: `IMAlgo`

Input: A complete graph $K = \langle S, E \rangle$.
Two non-negative integers n and k.
A relational opertor op is one of $\{=, <, >, \leq, \geq\}$.
Output: A set S_1 of at most n answer sets whose distance is $d_s(S_1)$ op k.

$E' \leftarrow \{\{s_j, s_i\} \mid s_j \neq s_i$ denote $s_j, s_i \in S, d(s_j, s_i)$ op $k\}$
$P = \{S\}$
$R = \{\}$
$X = \{\}$
S_1 = **BronKerbosch**(P, R, X, E')

Algorithm 3: `BronKerbosch(`P, R, X, E'`)`

if $P \cup X = \emptyset$ **then**
  if $|R| \geq n$ **then**
    pick R as a maximal clique of size at least n
    $S_1 \leftarrow$ report the first n elements of R
    Output S_1
    terminate
choose a pivot $u \in P \cup X$
for *each vertex* $v \in P \backslash nbrs(u)$ **do**
  BronKerbosch($P \cap nbrs(v)$, $R \cup v$, $X \cap nbrs(v)$)
  $P \leftarrow P \setminus v$
  $X \leftarrow X \cup v$

Algorithm 4: `MIMCase2`

Input: The answer sets (AS)
Two non-negative integers n and k.
A relational opertaor op is one of $\{=, <, >, \leq, \geq\}$.
Output: A set S_1 of at most n answer sets whose distance is $d_s(S_1)$ op k.

$S \leftarrow$ Define a set of $|AS|$ vertices
$E \leftarrow \{\{s_j, s_i\} \mid s_j \neq s_i$ denote $s_j, s_i \in S, d(s_j, s_i)$ op $k\}$
S_1 = **IMAlgo**($\langle S, E \rangle$, n, k, op)

Figure 1.Algorithms

Case 2 Modified Interactive Method (MIM)

Instead of building a graph $K = \langle S, E \rangle$ of all the answer sets (nodes S) in the preprocessing method and then a clique is picked with specific distance value in the interactive method, we can build a complete subgraph (clique) with only the edges with the specific distance value that the user specifies. Our intuition behind building a clique during the interactive method which is so-called modified interactive method is to present only to the user the answer sets (AS) he is interested in. Thus, we save the memory since the complete graph K is not built in the preprocessing method. Additionally, the execution time is reduced of the preprocessing method. Algorithm 4 shows the modified interactive method case.

The inputs of the above algorithm are the answer sets (AS) that the user specifies, the two non-negative integer numbers n and k; n is the number of the answer sets that differ in k atoms, and a relational operator op with $op \in \{<, >, \leq, \geq, or =\}$. At first, a complete undirected graph K_m is built from AS whose nodes are S and the edges E are labeled by the distance between pairs of nodes. Then, the IMAlgo algorithm is invoked with specific arguments, such as, the complete graph K_m, n, k, and the relational operator op, to find the maximal clique and return a set S_1 of at most n answer sets whose distance is $d_s(S_1)$ op k. The interactive method and the modified interactive method are different only in the inputs.

NAVAS Tool

In this section, we describe the graphical user interface of Navigation Approaches for Answer Sets (NavAS) and explain step by step how to use the tool.

Description

The start window of NavAS tool is as depicted in Figure 2.

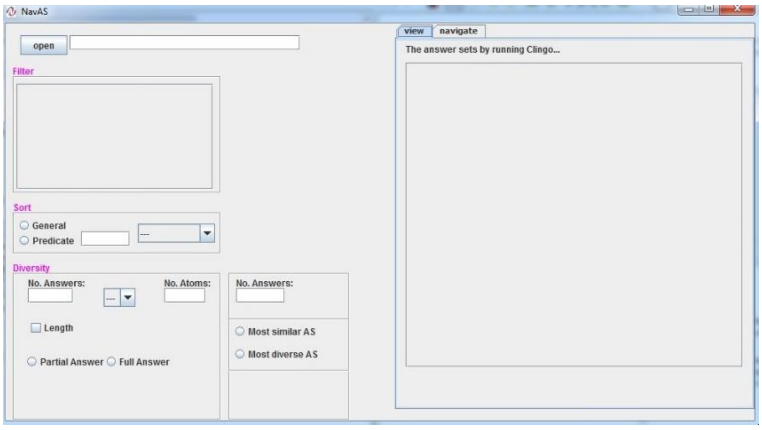

Figure 2. NavAS user interface

The starting point is to select a file with an *.lp* extension by the open button which is at the left top corner. Clicking on this button brings up a dialog box allowing to browse for the data file on the local file system. Once the user clicks on the open button, the answer sets as a result of running the Clingo solver, are displayed in a scrollable text area in the tab view in the right top corner.

Diversity Box

The box contains many components to find the diversity between the solutions. (see Figure 3). To illustrate, there are three components available on this box:

- ***No. Answers***. NavAS allows the user to type the number of solutions that he wants to show.
- ***No. Atoms***. The number of atoms that the solutions are different in.

The working scenario of the box is as following: "Show me the answer sets (No. Answers) that are different in one of $\{=, <, >, \leq, \geq\}$ of (No. Atoms)."

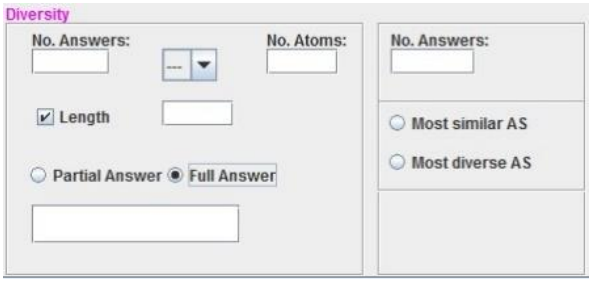

Figure 3. Diversity box

There is a length option which is applied when the solutions have different number of ground instances. The user can decide which answer set he wants as a pivot to find the diversity by either typing the full answer set or a part of it. To the right of the diversity box is the box for finding the most similar and diverse answer sets. The number of answer sets specifies by the user (No. Answers).

Evaluation

In this section, we make an evaluation of the performance of the NavAS tool. We consider a pizza configuration. We discuss the experiment of result on this example. We ran the experiment on an Intel machine with processor

2.30 GHz Intel Core i5 and memory 4 GB 665.1 MHz DDR3.

Pizza Example

This example is implemented in ASP to generate different pizza configurations. The input of the ASP program are the toppings with their categories to specify the type of pizza and the price of each of them. To accomplish this task, several rules and constraints on toppings have to be satisfied. Thus, a different number of toppings provide many different configurations of pizza.We consider herein a sample of the code to explain the facts of the program as input. In the following, we give a listing of some facts of the program (Figure 4).

```
price(bacon,110).
has_category(bacon,meat).
price(chicken,120).
has_category(chicken,meat).
price(peas,140).
has_category(peas,veg).
```

Figure 4. Facts of the program

We have a topping bacon which is indicated under meat category, and its price is `price(bacon,110).` The same thing for other facts. We add many facts to the pizza program to increase the search space (Figure 5).

```
Answer: 1

on(dough) on(tomato_sauce)
on(mozzarella) on(oregano) on(bacon)
on(broccoli) on(caper) total(870)
normal

Answer: 2

on(dough) on(tomato_sauce)
on(mozzarella) on(oregano) vegetarian
on(caper) on(basil) total(750)
```

Figure 5. Code for increase the search space

Usually there are too many answer sets of pizza example computed by an ASP solver. The user needs to compare these answer sets, by analyzing the similar/diverse ones with respect to some distance measure. To this end, we use the tool on this example to evaluate the performance of it. For similarity and diversity, we do experiment for testing the computation time of finding diverse/similar solutions. We measure the execution time of the preprocessing and the interactive method. We take five samples for each value and compute the average of them.

We consider several parameters to assess the performance of the tool with respect to finding similar/diverse answer sets:

(1) number of answer sets, (2) preprocessing method, (3) (modified) interactive method. Table 1, 2, 3, and 4 report the time of the case of interactive and modified interactive method with different values of k (number of different atoms), $k = 1, 2$, respectively, and the number of solutions n (the maximum size of the maximal cliques corresponding to the value of k).

From Table 1 and 2, we note that the execution time for the preprocessing method is quite high because of the storing and building a complete undirected graph K. We can see that the execution time for the number of answer sets 2000 is higher than 1000. For the number of atoms that the answer sets are different, we note that the execution time for $k = 2$ is higher than the execution time for $k = 1$. In fact, this depends on the configurations of the answer sets of a problem, the distance between them, and the ordering of the answer sets in the complete graph K.

From Table 3 and 4, we note that the execution time for the preprocessing method is less than the execution time for the preprocessing method in the previous tables. The preprocessing method herein is only used to store the answer sets. The graph K is built during the modified interactive method with a specific answer sets that the user specifies.

Table 1. Time execution for $(k = 1 \text{ and } n = 3)$ for pizza example

ANSWER SETS (AS)	*PREPROCESSING (MINUTES)*	*INTERACTIVE METHOD CASE1 (MINUTES)*
1000 (137KB)	0.5441	0.0046
2000 (278KB)	1.3237	0.0150

Table 2: Time execution for $(k = 2 \text{ and } n = 8)$ for pizza example

Answer Sets (AS)	*Preprocessing (minutes)*	*Interactive Method Case 1 (minutes)*
1000 (137KB)	0.5441	0.0107
2000 (278KB)	1.3237	0.0505

Table 3: Time execution for $(k = 1$ and $n = 3)$ for pizza example

Answer Sets (AS)	*Preprocessing (minutes)*	*Modified Interactive Method Case 2 (minutes)*
1000 (137KB)	0.4669	0.0525
2000 (278KB)	0.7868	0.4039

Table 4: Time execution for $(k = 2$ and $n = 8)$ for pizza example

Answer Sets (AS)	*Preprocessing (minutes)*	*Modified Interactive Method Case 2 (minutes)*
1000 (137KB)	0.4669	0.0454
2000 (278KB)	0.7868	0.4073

In general, we note that the execution time for the modified interactive method case is higher than the execution time for the interactive method in the previous tables, because the time needed to build the graph K is included in the execution time for the modified interactive method.

Conclusion

There is a large number of the answer sets available of an ASP program, all of them are not of the user's interest. Thus, a navigation of the search space could be a solution to help the user to access a specific answer sets. We studied finding similar/diverse solutions of the answer sets. We offered scenarios to find similar/diverse solutions. To this purpose, we introduced preprocessing and interactive methods and applied some distance measures. Regarding practical use, we presented NavAS, a tool for navigating the answer sets for general ASP programs. Finally, we made an evaluation of the performance of NavAS tool with Pizza example. We recorded the execution time for finding diverse/similar answer sets.

Scientific Ethics Declaration

The author declares that the scientific ethical and legal responsibility of this article published in EPSTEM journal belongs to the author.

References

Afeefi, A. (2019, April). NavAS: Navigation Approaches for Answer Sets. In *2019 IEEE Jordan International*

Joint Conference on Electrical Engineering and Information Technology (JEEIT) (pp. 79-84). IEEE. https://doi.org/10.1109/JEEIT.2019.8717370

Ambroz, T., Charwat, G., Jusits, A., Wallner, J. P., & Woltran, S. (2013, September). ARVis: Visualizing relations between answer sets. In *International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 73-78). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40564-8_8

Baum, D. (2004). *Finding All Maximal Cliques of a Family of Induced Subgraphs*., 1–15. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.358

Bron, C., & Kerbosch, J. (1973). Algorithm 457: Finding all cliques in an undirected graph, Community. *ACM*, *16*(9), 575-577. https://doi.org/10.1145/362342.362367

Busoniu, P. A., Oetsch, J., Pührer, J., Skočovský, P., & Tompits, H. (2013). SeaLion: An eclipse-based IDE for answer-set programming with advanced debugging support. *Theory and Practice of Logic Programming*, *13*(4–5). https://doi.org/10.1017/S1471068413000410

Computing the Stable Model Semantics. (n.d.). Retrieved June 30, 2021, from http://www.tcs.hut.fi/Software/smodels/

Das, A., Sanei-Mehri, S.-V., & Tirthapura, S. (2020). Shared-memory parallel maximal Clique Enumeration from static and dynamic graphs. *ACM Transactions on Parallel Computing*, *7*(1), 1–28.

Deng, F., Siersdorfer, S., & Zerr, S. (2012). Efficient Jaccard-based diversity analysis of large document collections. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/2396761.2398445

Eiter, T., Erdem, E., Erdoğan, H., & Fink, M. (2009). Finding similar or diverse solutions in answer set programming. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5649 LNCS*. https://doi.org/10.1007/978-3-642-02846-5_29

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of answer set programming. *AI Magazine*, *37*(3), 53–68.

Everardo, F., & Osorio, M. (2020). Towards an answer set programming methodology for constructing programs following a semi-automatic approach – extended and revised version. *Electronic Notes in Theoretical Computer Science*, *354*, 29–44.

Fakecineaste : How to Calculate the Hamming Code. (n.d.). Retrieved June 30, 2021, from http://fakecineaste.blogspot.com/2012/11/how-to-calculate-hamming-code.html

Fandinno, J., & Schulz, C. (2019). Answering the "why" in answer set programming - A survey of explanation approaches. *Theory and Practice of Logic Programming*, *19*(2). https://doi.org/10.1017/S1471068418000534

Febbraro, O., Reale, K., & Ricca, F. (2011). ASPIDE: Integrated development environment for answer set programming. In Logic Programming and Nonmonotonic Reasoning (pp. 317–330). Berlin, Heidelberg: Springer Berlin Heidelberg.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). *Clingo = ASP + Control: Preliminary Report*. *1*. http://arxiv.org/abs/1405.3694

Güven, Ç., & Atzmueller, M. (2019). Applying answer set programming for knowledge-based link prediction on social interaction networks. *Frontiers in Big Data*, *2*, 15.

iGROM download | SourceForge.net. (n.d.). Retrieved June 30, 2021, from https://sourceforge.net/projects/igrom/

Li, Y., Shao, Z., Yu, D., Liao, X., & Jin, H. (2019). Fast maximal clique enumeration for real-world graphs. In Database Systems for Advanced Applications (pp. 641–658). Cham: Springer International Publishing.

Makino, K., & Uno, T. (2004). New algorithms for enumerating all maximal cliques. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3111*. https://doi.org/10.1007/978-3-540-27810-8_23

Marcopoulos, E., Reotutar, C., & Zhang, Y. (2017). *An Online Development Environment for Answer Set Programming*. *1*. http://www.math.ttu.edu/texprep/

Niemelä, I., & Simons, P. (n.d.). Smodels: An implementation of the stable model and well-founded semantics for normal lp".

Son, T. C., & Balduccini, M. (2018). Answer set planning in single- and multi-agent environments. KI - *Künstliche Intelligenz, 32(*2–3), 133–141.

Sureshkumar, A., De Vos, M., Brain, M., & Fitch, J. (2007). APE: An AnsProlog* environment. *CEUR Workshop Proceedings*, *281*.

Zhu, Y., & Truszczynski, M. (2013). On optimal solutions of answer set optimization problems. In *Logic Programming and* Nonmonotonic Reasoning (pp. 556–568). Berlin, Heidelberg: Springer Berlin Heidelberg.

Author Information

Asmaa AFEEFI
An-Najah National University
PO. Box 7
Nablus, West Bank, Palestine
Contact e-mail: asmaafeefy@najah.edu

To cite this article:

Afeefi, A. (2021). IMSD: Interactive methods for finding similar or diverse answer sets. *The Eurasia Proceedings of Science, Technology, Engineering & Mathematics (EPSTEM), 12*, 85-94.