Analysis of ASEO Approximation Algorithms for PASP Likelihood Inference

Yuri Simantob
Universidade de São Paulo, ETH Zürich
ysimantob@{usp.br, student.ethz.ch}

Denis Deratani Mauá
Universidade de São Paulo
ddm@usp.br

November 29, 2024

1 Abstract

Until now, the inference of queries to probabilistic logic programs (PLP) has been limited to small-scale examples due to the brute-force nature of the inference algorithms, enumerating all potential solutions. Thus, the use of Answer Set Enumeration by Optimality (ASEO) algorithms for approximate inference was experimented with, and an initial version implemented into the dPASP framework. This report contains the results of experiments analyzing differences in performance between stratified and non-stratified programs. While no notable differences could be found, large variations in the convergence dynamics were witnessed, affecting the predictability and usability of an approximate inference algorithm in dPASP.

2 Introduction

This research project aims to measure the performance of an algorithm that approximates the probability of a specific outcome in a probabilistic logic program (PLP). A probabilistic logic program is nothing else but a set of probabilistic logic rules describing a universe of logical atoms, which can then be used to compute probabilities about logical atoms in this universe. Until now, only experiments for exact algorithms have been conducted, i.e. algorithms that calculate the exact probability of such an outcome, which can become very costly for bigger PLP's. In these cases, approximating such probabilities can become indeed very attractive. The approximation algorithm we will be analyzing is based on the Answer Set by Enumeration (ASEO) approach [1], and is already

implemented into the dPASP framework/language, which will be used to conduct the experiments in. dPASP is a framework, which allows us to write and infer probabilities in PLP's. [2] In this report, we wrote 6 different PLP's, 3 stratified and 3 non-stratified, and analyzed the ASEO algorithm's differing performance to approximate the probabilities across these 6 experiments.

3 Recap

3.1 Logic Program

A logic program P is a set of logical disjunctive rules that are of the form $h \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$, where the head h can be trivial, making the negation of the body b act as a constraint in the program.

3.2 Probabilistic Logic Program

A probabilistic logic program (PLP), is a set of rules of a similar form $p :: h \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_m$, where p determines a probability between 0 and 1 that h will be true, given that the body is true.

3.3 Total Choice

A total choice θ of a probabilistic logic program P is its conversion into a standard logic program P_θ by choosing the outcomes of its probabilistic components. Each total choice θ uniquely determines a standard logic program P_θ and every PLP has a set of possible total choices, denoted by Θ.

3.4 Stratified Logic Program

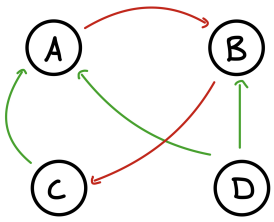
A stratified logic program is a logic program P with the very useful property, that it has only one unique minimal model.

3.4.1 Dependency Graph

Now, we can construct a dependency graph as follows. Each atom $a \in A$ in P is represented as a node, whereas for each positive head-body pair $h \leftarrow b_i$ there is a positive edge from b_i to h just as there is a negative edge of the same manner for each negative head-body pair $h \leftarrow \neg c_j$.

3.4.2 Stratification

A logic program is stratified if and only if all of its dependency graph's cycles do not contain any negative edges.



(a) Dependency Graph

```
a :- c, d.
b :- not a, d.
c :- not b.
```

(b) Example Program

Figure 1: The dependency graph of a stratified program

4 The dPASP framework

The aim of dPASP is to provide a framework that integrates probabilistic components and support for neural networks, giving the user the possibility to infer probabilities in data-driven scenarios.[2] The underlying inference is mainly based (1) on the listing of total choices $\theta \in \Theta$, (2) the inference of satisfaction for each logic program P_θ corresponding to one of the total choices and (3) the aggregation of the obtained inferences given the probability $\mathbb{P}(\theta)$ of each total choice. (1) and (2) is done using the clingo solver developed by the Potsdam Answer Set Collection, while (3) is calculated as following:

$$\mathbb{P}(Q) = \sum_{I:\theta\in\Theta} \mathbb{P}(\theta)\frac{N_{I\models Q}}{N_I} \quad (4)$$

N_I is the **number of models of the total choice** θ and $N_{I\models Q}$ is the number of models of θ **where the query** Q **is satisfied**.

As one can see, such an inference technique requires the enumeration of all models N_I for each total choice θ. However, for a growing number of atoms in a logic program, the space of total choices Θ may increase exponentially, which limits the applicability of dPASP's inference capabilities to small-scale examples that are mainly used for demonstration purposes.

4.1 The ASEO algorithm and approximate inference

As a response to this problem, a new inference method was implemented into the dPASP source. Since the underlying clingo solving relies on grounding (gringo) and the actual solving (clasp), the idea was to use an ASEO algorithm [1], which would bound the total number of considered models, i.e. the sum of all N_I and enumerate the answer sets by a certain optimality criterion. Gringo supports the grounding of a select total choice θ and clasp has support for minimization techniques determined by weak rules, allowing for an ASEO implementation in clingo (for more info, see [3]). This is what we call **approximate inference** with a model bound B in contrast to **exact inference** as shown in (4).

4.1.1 The Optimality Criterion

The optimality criterion in play here is essentially based upon the base probability $\mathbb{P}(\theta)$ of each total choice. Thus, the first models considered are the one's belonging to the total choice θ_1, which has the highest probability out of all total choices, i.e. $\mathbb{P}(\theta_1) \geq \mathbb{P}(\theta)$ for $\theta \in \Theta$. Then the models belonging to θ_2, where $\mathbb{P}(\theta_2) \geq \mathbb{P}(\theta)$ for $\theta \in \Theta \setminus \theta_1$ are enumerated, and so on until the bound of enumerated models B is reached. The higher we choose such B, the more likely it is that we get a good approximation of the exact probability $\mathbb{P}(Q)$, but the lower we choose B, the less computation we need to ground and solve the resulting logic programs.

4.2 The Issue of the ASEO algorithm

While the implementation of an ASEO algorithm for approximate inference as in [1] into the dPASP framework is a good start to facilitate inference for complex scenarios, there prevails the underlying issue of **enumeration by the most probable total choice** that may give rise to very differing approximation performances, depending on the nature of a logic program. More specifically it can very well happen that the i most probable total choices may badly sample the actual distribution of models satisfying the query Q. A very forced, but telling example would be the following program:

```
0.49 :: c.

a :- not b, c.
b :- not a, c.

#query(a)
```

Which has exactly 2 models for $c = 1$ (1 has a true and one doesn't) and 1 model for $c = 0$ (where a is false).

Now the total choice with the highest probability is the one where $c = 0$, so if we selected our model bound B to be exactly 1, our approximated result for $\mathbb{P}(Q) = \mathbb{P}(a)$ would be 0. We would need to increase the model bound here to 2 or even 3 in order to get closer to our exact probability $\mathbb{P}(a) = \frac{0.49}{2}$, which in this case would already make up more than half of the total models for the entire probabilistic logic program.

4.3 The Link to Stratification

Although the example is very primitive, we suspect that this property of the algorithm could present an issue when presented with much more complex logic programs. Consequently, we will try quantify and compare the approximation performance of ASEO inference for various examples of PLP's grouped into two categories: **stratified** and **non-stratified** logic programs. The aim is to

answer the question whether there exists a substantial amount of differing ASEO performance between both categories.

The reason why we suspect ASEO to perform unequally well on **stratified** logic programs is the fact that each total choice has only one model to solve for. So whenever we allow ASEO to run B models, we automatically let it run B total choices too, and thus it makes the enumeration of models/total choices by the probability of a total choice $\mathbb{P}(\theta)$ much more reasonable. For **non-stratified** programs we could end up in a situation where the most probable (but maybe still absolutely unlikely) total choice θ_1, could use up all of our B models we want to enumerate, while a slightly less likely $\theta_i,\ i > 1$ may have much less than B models and ASEO theoretically could search for rest of the models in other total choices $\theta_j, \theta_k, \ldots$ whose total probability may sum up to much more than the one of θ_1.

4.3.1 Weight Imbalance

In short, when we solve for models using a given total choice, ASEO weights the importance of such a model based on the probability of the entire total choice, and not based on the importance of the model inside the total choice itself. For a total choice with 2 models, one model has a much stronger contribution to the final inferred probability than in an equally likely total choice with 1000 models, and ASEO wouldn't account for such a weight imbalance. Seeing that such a weight imbalance doesn't exist for **stratified** logic programs, we bring the assumption forward that ASEO approximate inference performs better for stratified programs than it does for non-stratified ones. The aim of the following experiments serve to lay a foundation to qualitatively analyse the convergence dynamics in the different scenarios and detect differences in convergence between stratified and non-stratified programs.

5 Experiments

5.1 Terminology

- Q_P: A query Q to a (probabilistic) logic program P
- B: The set of model bounds. ASEO will use each B_i for an inference
- $E(Q_P, B_i)$: The approximation error of the ASEO inference of a query Q to a logic program P given a model bound B_i
- $\mathbb{P}_{\text{exact}}(Q_P)$: The exact probability of a query Q to a logic program P
- $\mathbb{P}_{\text{aseo}}(Q_P, B_i)$: The approximated probability ASEO calculated for Q and P given a model bound B_i
- B_0: The smallest model bound we give ASEO (in our case we chose $B_0 = 2$)

5.2 Our programs

Firstly, we set out to quantify the performance of our ASEO implementation in dPASP. As a base for comparison, 6 different probabilistic logic programs were written, 3 of them stratified and 3 of them non-stratified.

5.2.1 Non-Stratified Programs

1. Graph Coloring Problem
2. Latin Square Problem
3. Argumentation Scenario [4]

5.2.2 Stratified Programs

1. Smoke-Stress Problem
2. Mango Selling Problem
3. Summing Arithmetic Expressions

All of these problems have the property in common, that whenever we modify the number of atoms, the number of models grows exponentially fast, making each of them relevant for approximate inference (see Figure 2).

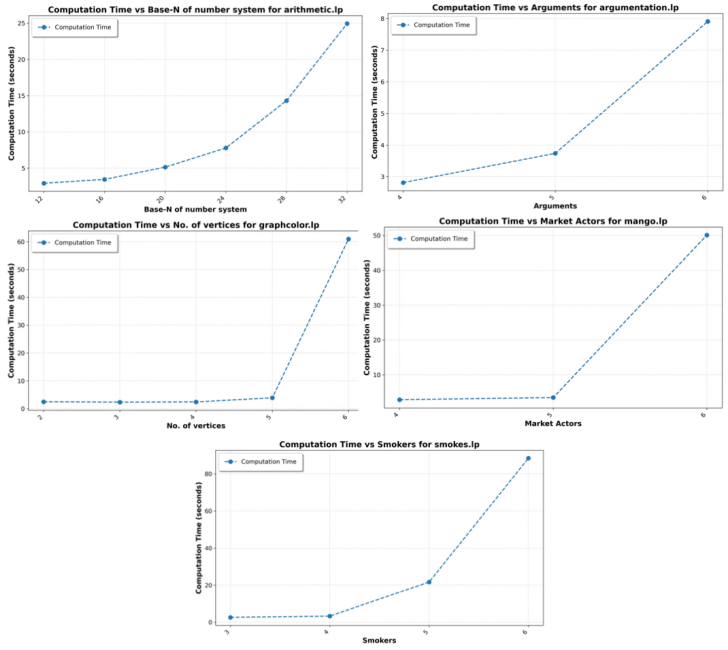


Figure 2: Runtimes for the Experiments given different starting parameters (Note: The Latin Square experiment couldn't be run with more parameters due to enormous runtime increases)

5.3 Choice of Hyperparameters

In order to make a fair comparison between the 6 experiments we must ensure that they are roughly of equal size. For this, the experiments have been re-written such that their number of emitted models lies in the interval of $[40,000, 400,000]$ models. Regarding the error metric we will consider the number of models logarithmically only, so this is a relatively small interval, also considering that the number of models our experiments emit increases exponentially, the more complicated we make the experiments (as in Figure 2).

Since the number of models still vary, we also want to dynamically create the set of model bounds B adjusted for the number of models a certain experiment emits. We choose $B_0 = 2$ and B_n to be the number of models the experiment emits. There are two methods that came to mind were either to linearly space the model bounds in the set (i.e. $[2, 1,000, 2,000, 3,000]$ for $B_n = 3,000$ and 4 experiments) or to logarithmically space them (i.e. $[2, 4, \ldots, 2^{40}]$ for $B_n = 2^{40}$ and 40 experiments).

The problem when linearly spacing the model bounds is that we don't really

capture the improvements when using small model bounds, which are of most interest to us when approximating big programs. Yet logarithmically spacing them gives us a too big of a focus on the random fluctuations in the beginning and doesn't capture the convergence dynamic that well. So the solution we used was to increase the model bounds with a decreasing multiplicative factor between the elements, a trade-off between the two previous methods.

One key assumption we are making throughout the experiments is that the number of models a certain PLP emits is directly proportional to the runtime the inference needs. So even though the direct interest of this analysis is to get a quantification of how much time one can save for a certain approximation error margin, we chose our experiments to be equal in model size, rather than in inference time.

5.4 The Metric

The process of designing the metric had to overcome the following issues

1. The convergence was generally non-monotonous
2. Some experiments randomly had very good predictions in the beginning, whereas other initial predictions started off from far
3. In most experiments, the errors fluctuated unpredictably before the downward convergence to 0 began

In essence what we would like to know is how fast the approximation error decreases when increasing the amount of models enumerated. One naïve way of doing this would be to take the error curve over a range of model bounds B_i and then calculate the change of the error curve as B_i increases, i.e. calculate the derivative of E w.r.t. the model bound B.

There are some issues with this however. First of all, the derivatives give us a multitude of values back, and not a single value that captures the entire convergence dynamic of the algorithm well. We could take the average of course, but we would lose a lot of information to accurately ascribe a faster convergence to a certain program. This is why we opted for the **Area Under the Error Curve** (AUEC) metric, since we can then directly infer that a larger such value corresponds to a slower convergence rate and vice versa.

The biggest issue however was lying in the choice of designing the error metric we would then calculate the AUEC from. One first idea was to take the error E_0 of the smallest model bound (in our case $B_0 = 2$) and then normalize the other errors w.r.t. E_0.

$$\text{SMBE}(Q_P, B_i) = \frac{\mathbb{P}_{\text{exact}}(Q_P) - \mathbb{P}_{\text{aseo}}(Q_P, B_i)}{\mathbb{P}_{\text{exact}}(Q_P) - \mathbb{P}_{\text{aseo}}(Q_P, B_0)}$$

We will call this error the Smallest Model Bound Error (SMBE) from now on. The reason for this choice was to eliminate the distortion good or bad initial predictions cause in the error metric. Now if the convergence would be monotonous this may have been a great choice, yet it was not, and consequentially we could witness large increases in error for small B_i that could give us unreasonably large error values for some fluctuations in the beginning (see Figure Y).

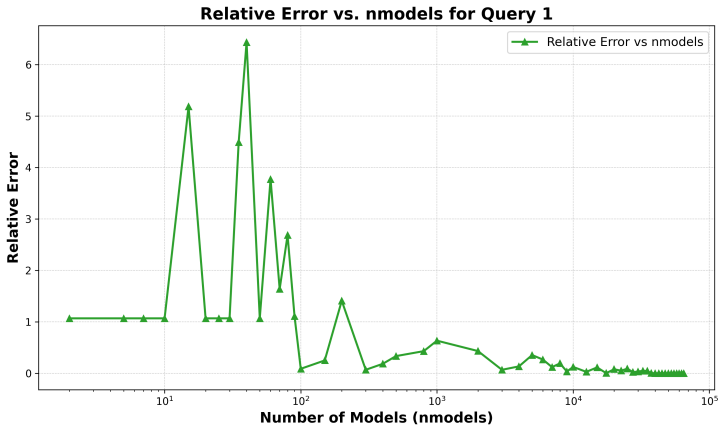


Figure 3: Error of Inferring the Latin Square Program using the Smallest Model Bound Error

The next idea was to normalize the error w.r.t. the maximum error, but then, in the case of the Latin Square experiment, it would happen that most of the errors remained artificially small, because there were some relatively large fluctuations for small B_i. So we chose to normalize the error w.r.t. the average of the maximum error and the Smallest Model Bound Error:

$$\text{maxerror} = \max([E(Q_P, B_j)]_{j=0}^{n})$$

$$C = \frac{\text{maxerror} + \text{SMBE}(Q_P, B_i)}{2}$$

$$E_2(Q_P, B_i) = \frac{E(Q_P, B_i)}{C}$$

With this choice we balance out the effects non-monotonicity and initial random fluctuations have on our error metric a bit. In order to further reduce the effect of initial random fluctuations we log-scale the errors obtained:

$$E_{\text{metric}}(Q_P, B_i) = \log(1 + (e-1) \cdot E_2(Q_P, B_i))$$

Eventually, this became the final error metric we used for calculating the AUEC values for each experiment.

5.5 The 6 experiments

The 6 experiments that were written (or modified) for this experiment were intentionally made to be primitive demonstrations of potential use cases of the dPASP framework. Ranging over pure combinatorial problems (Latin Square, Graph Coloring Arithmetic Number Building) to economic simulations (Mango Market) and social simulations (Argumentation Scenario Smoking Influence), they present a small glimpse of what mechanisms could be implemented with dPASP for more complex scenarios involving neural networks.

5.5.1 Mango Market

The mango market problem simulates a market where there are buyers and sellers. Sellers can sell mangos and apples but buyers come to this market to buy mangos only. What's more, the buyers are very picky and only buy from sellers which exclusively sell mangos. Sellers in turn probabilistically have either apples, mangos or both at their booth. To capture the complexity of this dynamic we want to know what the probability is that a deal happened at the market.

5.5.2 Argumentation Scenario

Here we want to simulate the dynamics of a debate. Given is a set of arguments A and a graph $G = (A, E)$ connecting these arguments. In this graph, there is a directed edge from argument u to argument v if and only if v can logically follow from u. The weight of the edge determines the probability that a debater will follow up with v after saying u. $F \subset A$ is a subset capturing a fact. A debater wins the debate if he reaches all $a \in F$ and his opponent does not. Both debaters begin with two distinct arguments a_1 and a_2. Based on this, we want to know what the probabilities are that one or the other debater win the debate.

5.5.3 Smoking Dynamics

Given here are a set of persons P, where a subset $S \subset P$ of them smoke. A non-smoker can become a smoker either if he/she becomes stressed (which happens at a certain probability), or if a friend of this person smokes. We've set a social network and the initial smokers and we want to find out what the probability is that a certain non-smoker begins to smoke.

5.5.4 Graph Coloring Problem

The graph coloring problem is a classical problem in Computer Science and graph theory, where we don't want two neighboring vertices to share the same color. Our PLP creates a random graph, uses this random graph to go through all possible color combinations of the graph and gives us the probability of a certain color combination given a valid coloured graph.

5.5.5 Latin Square Problem

The Latin Square problem is a classical combinatorial challenge used to demonstrate huge growing complexity. The goal here is to fill a $n \times n$ grid with digits such that each digit appears exactly once in each row and column. For your reference, the aim of a sudoku is to solve a 9x9 Latin Square problem (with some digits already filled in). Our Probabilistic Logic Program tests all the possible combinations and tells us how many of these combinations make up a valid Latin Square.

5.5.6 Arithmetic Number Building

Here we build a random number in a N-ary number system, by uniformly sampling for each digit. We then aggregate 3 digits to turn them into a number and query the probability that this number was lies between a certain interval.

6 Results

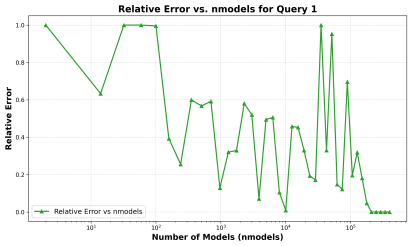
Running the 6 different programs yielded the following results:

Table 1: The resulting AUEC's given the experiments and their hyperparameters

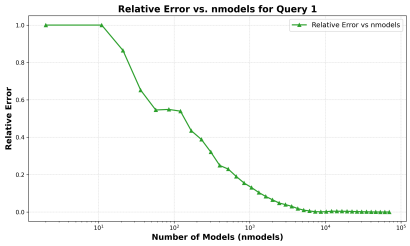
Experiment	**AUEC**	**Models**	**Scale Description**	**Stratified?**
Latin Square	4.406	45,000	2x2 square	Yes
Mango	6.716	200,000	4 vendors, 3 buyers	No
Graph Coloring	7.233	400,000	5 vertices	No
Smoke-Stress	4.708	200,000	7 people	Yes
Arithmetic	7.060	140,000	25-ary number system	Yes
Argumentation	4.641	70,000	8 arguments	No

Given the low number and the very specific nature of the experiments, a rigorous statistical test to reject or accept the hypothesis that stratified programs converge faster than non-stratified programs is virtually impossible. What can be seen here is that the average AUEC values are slightly higher for the non-stratified programs than the stratified programs. Yet this doesn't even come close to indicating a possible correlation between stratification and faster convergence times.

More interesting is to look at the shape of the error curve given the model bounds B. Some experiments showed very steady convergence, whereas others fluctuated violently until shortly before $B_i = B_n$. Here are two examples in contrast:



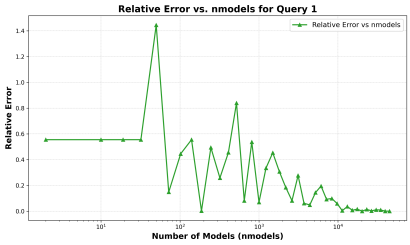
(a) E_{metric} of the Graph-Coloring Program



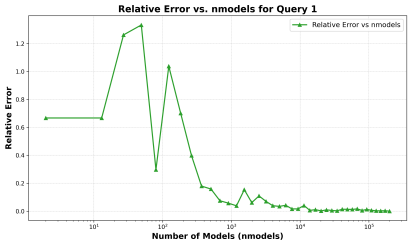
(b) E_{metric} of the Argumentation Program

Figure 4: Stark Contrasts in Convergence Dynamics

Note that both programs are non-stratified. Similar, less stark contrasts can be made within the stratified programs:



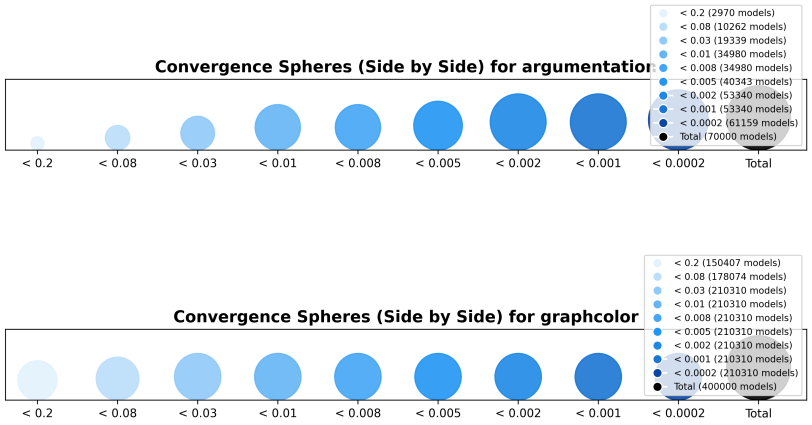
(a) E_{metric} of the Arithmetic Program



(b) E_{metric} of the Smoke Program

Figure 5: Contrasts in Convergence Dynamics (Stratified)

Here we can also see that there is no big difference of convergence smoothness between stratified and non-stratified programs. Problems with smooth error convergence curves are optimal for ASEO usage because they give us more confidence in the expected error margin we get from using a certain number B_i of models for our ASEO inference. Here is another graphical view of the number of models B_i we needed to get constantly below a certain error threshold:



(a) Graphical Representation of the minimal amount of models needed to attain $E_2 < \tau$

Here we can see exactly that. The number of models that give us a $E_2 < 0.2$ guarantee in the case of the argumentation program is very small compared to the total amount of models whereas in the graph-coloring program almost half of all the models are required.

Experiment Name	$\tau < 0.2$	$\tau < 0.01$	$\tau < 0.0002$
Argumentation	4.24%	**49.97%**	76.20%
Arithmetic	12.90%	**100%**	**100%**
Mango	17.2%	54.34%	74.14%
Graph Coloring	**37.60%**	52.57%	**52.57%**
Smoke-Stress	**0.18%**	54.53%	**100%**
Latin Square	8.25%	77.08%	87.86%

Table 2: E_2 margin guarantees with the percentage of the total models used for ASEO inference

Regarding our assumption in Section 5.3. we can also see that the runtimes scale linearly with the amount of models used, so the values in Table Z can be directly used when talking about runtime-error trade-offs.

7 Discussion

As already hinted at in Section 5, our results do not indicate any difference in convergence speed between the stratified and non-stratified programs. Also regarding the convergence dynamics, we can not jump to any conclusion regarding smoothness and predictability of convergence between these two categories. In essence, there has been no meaningful clue that suggests differing performance of ASEO between stratified and non-stratified programs.

However, disregarding stratification, it is interesting to see how the convergence dynamics of the experiments vary a lot. Similar convergence plots from different, smaller-scale experiments in [4] mostly show smooth convergence, but as we can see in Figure 4, ASEO is not always able to converge as nicely. The two experiments in Figure 4 are those that stand out here: While the argumentation program has a very smooth (and almost monotonic) progression for ASEO approximation errors, the graph-coloring program seems to come to very different results when adding some more models to its approximative calculation.

In the case of the wild convergence behaviour, the graph-coloring program emits, one could be led to believe that this complicated error metric we've crafted may have distorted the actual behaviour when B_i is close to B_n. Yet exactly the same behaviour can be seen when we look at Figure 7, showing the approximated and the exact probability of the query:

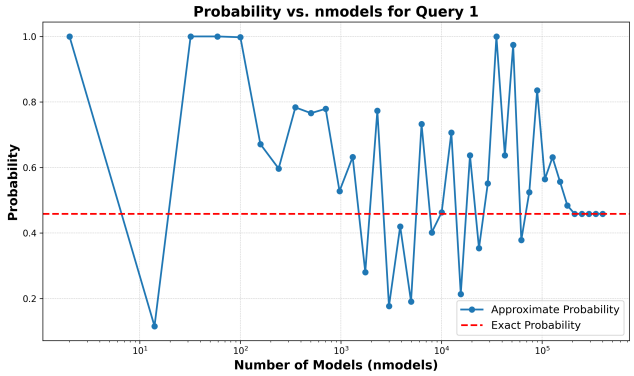


Figure 7: $\mathbb{P}_{\text{aseo}}(Q_{\text{graphcolor}}, B)$ vs. $\mathbb{P}_{\text{exact}}(Q_{\text{graphcolor}}, B_i)$

The numbers we obtained in Figure 2 give us the insight that we should be very careful when expecting good ASEO error margins for a model bound that is only a small fraction of the total models the program emits. Sure, being able to reduce the error from 1 to 0.2 with only $\frac{1}{500}$-th of the models looks promising but there are a few issues about getting too excited over this. Firstly, a 0.2 error bound is not of any use in practice, especially in a context when we want to combine the inference of multiple atoms. Secondly, it is somehow telling that

the best performance for a more feasible error ($\tau < 0.1$ already needs half of all the total models, which presents us with an almost negligible linear runtime speedup of 2. And lastly, we don't yet know how the error convergence would progress if we tried to make the smoke-stress program exponentially more complicated. We have no guarantee that for, say, a smoke-stress simulation with 20 people and a few billion models we could expect the same approximation error using $\frac{1}{500}$-th of these billion models.

8 Future Work

One thing this report can offer a bit better is an overview about the experiments that would reveal much more valuable insights into the convergence dynamics of dPASP's ASEO implementation. Obvious things first, we would gain much understanding of ASEO's practical usefulness if we would make this exact same analysis with much larger programs. Making these 6 experiments have a total number of models in the range of multiple millions or even billions instead of just a few thousands, would yield an interesting comparison to the results in this report, especially regarding smoothness of convergence. The groundwork for such an experiment has already been laid here, all the experiments/programs used in this report can be altered incredibly easy to be made larger and more complex (see Figure 2).

Regarding the Weight Imbalance issue discussed in Section 4.3.1, much could be learned from an altered implementation of ASEO, which would approximate the number of models a total choice θ emits in order to then weight the individual models more accurately. Such an approximation could be explored as proposed in [5] and be compared with the current one via a rigorous statistical test.

As soon as larger experiments have been conducted as suggested above, we can start building proto-practical experiments to reveal the error margin that one can work with. This would give us a much more accurate picture about how broad the use of an (ASEO) approximation algorithm could be and the runtime improvements we could achieve as compared to exact inference. The most interesting experiments would be the ones combining neural networks and probabilistic logic programming, exactly the kind of usage dPASP was made for.

9 Conclusion

Thanks to the specificity of the experiments, we at least know that there are some non-trivial PLP's out there that converge nicely. However, using ASEO inference for a dPASP program, one must not take smoothness of convergence

as a given and be very careful about error bound expectations.

The role approximate inference will play in the future of big probabilistic logic programming and its combination with neural networks to create AI applications will become a bit clearer if we scale up the size of such experiments and get some practical and useful applications to testing. This being said, our report doesn't diminish the potential approximate inference could have in the domain of complex and real-world-applied PLP's. The bottom line one can get out of the results is to remain cautious regarding one's expectations until the experiments of larger scale are conducted.

10 Information about the Author

This technical report is a project made for the course "Atividade Curricular em Pesquisa" (MAC0215) at the Mathematics and Statistics Institute of Universidade de São Paulo (IME-USP). The project was supervised by Denis Maratani Mauá and put into practice by myself, Yuri Simantob.

10.1 Required Information

- **NUSP:** 16246136

10.2 Timeline

	Month	Workload (h)
Literature Review	July-August	40
Familiarization with the Codebase	August-September	30
Writing the Experiments	September-October	15
Writing the Research Proposal	September-October	5
Testing & Plotting the Experiments	November	10
Aggregating Findings in the Report	November	10
Total		**110**

Figure 8: The Resulting Timeline over the Course of the Months

10.3 Codebase

The codebase can be found at: `https://github.com/famitzsy8/dPASP_ASEOPerformance`

References

[1] Jukka Pajunen and Tomi Janhunen. Solution enumeration by optimality in answer set programming. *Artificial Intelligence*, August 7 2021.

[2] Renato Lui Geh, Jonas Gonçalves, Igor Cataneo Silveira, Denis Deratani Mauá, and Fábio Gagliardi Cozman. dPASP: A comprehensive differentiable probabilistic answer set programming environment for neurosymbolic learning and reasoning. 2023.

[3] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Rühling, and Torsten Schaub. plingo: A system for probabilistic reasoning in clingo based on lpmln. *Artificial Intelligence*, September 2 2022.

[4] Renato Lui Geh, Jonas Gonçalves, Igor Cataneo Silveira, Denis Deratani Mauá, and Fábio Gagliardi Cozman. dpasp: A probabilistic logic programming environment for neurosymbolic learning and reasoning. 2024.

[5] M. Kabir, F. O. Everardo, A. K. Shukla, M. Hecher, J. K. Fichte, and K. S. Meel. Approxasp – a scalable approximate answer set counter. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 5755–5764, 2022.