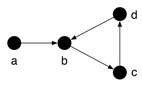
Difflog: Synthesizing Datalog Programs using Numerical Relaxation

Xujie Si*, Mukund Raghothaman*, Kihong Heo, Mayur Naik (*equal contribution)

Background and Motivation

Learning logical rules from relational input-output data

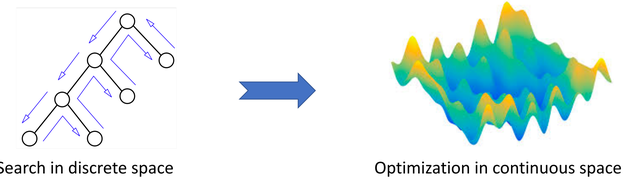


edge (input)	
a	b
b	c
c	d
d	b

scc (output)		
desired	b	c
undesired	a	d
desired	c	d

```
path(x, y) :- edge(x, y).
path(x, z) :- edge(x, y), path(y, z).
scc(x, y)  :- path(x, y), path(y, x).
```

- Traditionally learned using ***discrete combinatorial approaches***
- ***Gradient-based approaches*** are remarkably successful in machine learning.
- ***Can gradient-based approaches greatly help to learn logical rules?***



Search in discrete space → Optimization in continuous space

Importance of Learning Logic Programs

- Challenge problem in AI
- Good ***interpretability*** and ***compositionality***
- An ideal model to represent the learned knowledge
- Logic programs widely used in many areas

Big Data Analytics: DeepDive, LogicBlox, BigDatalog

Networking: RapidNet, SociaLite

Program Analysis: semmle, DOOP, PETABLOX, JChord

Challenges and Our Approach

- ❑ How to ***relax logical rules***?
- ❑ How to ***efficiently learn*** a relaxed program?
- ❑ How to ***recover*** classical program?

Key Ideas

- ❖ Associate each rule with a weight
- ❖ Each tuple will get a weight (depending on how it is derived)
- ❖ Turn combinatorial search into continuous optimization
- ❖ Recover the desired logic program from the optimized weights

Performance Optimization

Newton-Raphson Method — Focus on the exact program synthesis where zero loss is achievable

+

Periodic MCMC Sampling — Help to escape local minimum

+

Early Termination — Not ad hoc, a systematic approach based on a sound separation testing

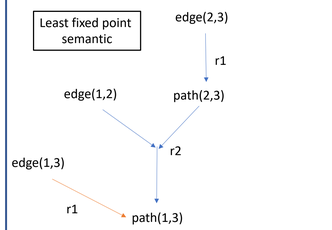
+

Parallelization — Leverage the fact of being a type of Las Vegas algorithm

Numerical Relaxation of Logic Programs

```
r1 path(x, y) :- edge(x, y).  // 0.7
r2 path(x, z) :- edge(x, y), path(y, z).  // 0.9
```

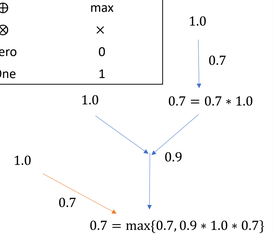
Least fixed point semantic



$$v_t = \bigvee_g (v_{a_1} \wedge v_{a_2} \wedge \cdots \wedge v_{a_k})$$

Logic world

Viterbi Semiring	
Set	[0,1]
$\oplus$	max
$\otimes$	$\times$
Zero	0
One	1



$$v_t = \max_g (w_g \times v_{a_1} \times v_{a_2} \times \cdots \times v_{a_k})$$

Numerical world

Easy to modify existing Datalog solvers with only $O(1)$ overhead.

$$L(w) = \sum_{t \in pos} (1 - v_t)^2 + \sum_{t \in neg} (0 - v_t)^2$$

$$w^{(i+1)} = w^{(i)} - L(w) \frac{\nabla_w L(w)}{\|\nabla_w L(w)\|^2}$$

Connections to Classical Datalog

Theorem: Each tuple weight v_t varies continuously and monotonically increases with the rule weights $\boldsymbol{w}$.

The loss surface is still NOT monotonic.

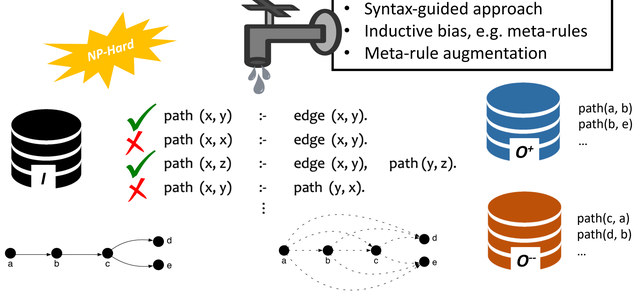
Theorem: Each tuple weight v_t is non-zero if and only if it can be derived in the classical program.

This enables us to recover classical program.

Rule Selection Problem

NP-Hard

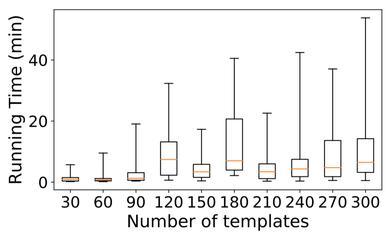
Rule generation approaches
- Syntax-guided approach
- Inductive bias, e.g. meta-rules
- Meta-rule augmentation



Empirical Evaluation Results

Benchmark	Rel	Rule Exp	Rule Cnd	Tuple In	Tuple Out	DIFFLOG Iter	DIFFLOG Smpl	DIFFLOG Time	ALPS Time
polysite	6	3	552	97	27	17	1	**27**	84
downcast	9	4	1,267	89	175	5	1	**30**	1,646
rv-check	5	5	335	74	2	1,205	41	**22**	195
andersen	5	4	175	7	7	1	0	**4**	27
1-call-site	9	4	173	28	16	4	1	**4**	106
2-call-site	9	4	122	30	15	25	1	**53**	676
1-object	11	4	46	40	13	3	1	**3**	345
1-type	12	4	70	48	22	3	1	**4**	13
escape	10	6	140	13	19	2	1	**1**	5
modref	13	10	129	18	34	1	0	**1**	2,836

Significantly outperform the state-of-the-art synthesizer



Scale to a large number of templates

Benchmark	Hybrid Best	Hybrid Median	Hybrid Timeout	Newton Best	Newton Median	Newton Timeout	MCMC Best	MCMC Median	MCMC Timeout
polysite	27s	142s	0	10s	72s	0	12s	76s	0
downcast	30s	310s	2	16s	252s	9	70s	268s	7
rv-check	22s	948s	2	N/A	N/A	32	N/A	N/A	32
andersen	4s	29s	0	3s	15s	10	4s	17s	9
1-call-site	4s	18s	0	8s	18s	1	N/A	N/A	32
2-call-site	53s	225s	0	27s	N/A	17	42s	94s	9
1-object	3s	17s	0	3s	N/A	17	N/A	N/A	32
1-type	4s	12s	0	3s	N/A	18	N/A	N/A	32
escape	1s	2s	0	1s	N/A	17	N/A	N/A	32
modref	1s	2s	0	1s	1s	4	N/A	N/A	32
Total			4			125			217

Hybrid optimization is essential to success

Contact: {xsi, rmukund, kheo, mhnaik}@cis.upenn.edu

University of Pennsylvania

https://github.com/petablox/difflog