

Extensions to the Relational Paths Based Learning Approach RPBL

Zhiqiang Gao, Zhizheng Zhang
Institute of Computer Science and Engineering
Southeast University, Nanjing, China

Zhisheng Huang
Department of Computer Science
Vrije Universiteit Amsterdam, The Netherlands

Abstract

In this paper we extend RPBL, a Relational Paths Based Learning approach for first order theories in three directions. We apply domain theories to expand structured instance space, learn recursive theories by an example of learning member relationship of lists, and analyze the performance as well as time complexity theoretically. In addition, we give the details of our experimental results.

1. Introduction

Inductive logic programming (ILP) approaches based on first-order representations have been successfully applied to relational learning. Standard ILP approaches usually take sequential covering algorithm for learning rule sets based on the strategy of learning one rule, removing the data it covers, then iterating this process, such as FOIL [Quillian 1990], FOCL [Pazzani et al. 1991], and Forte [Richards & Mooney 1991]. This sequential covering algorithm performs a greedy search, formulating a sequence of rules without backtracking. Greedy search relies on hill-climbing heuristics to avoid combinatorial explosion, and is susceptible to local maxima and local plateaus [Richards & Mooney 1992].

The earlier work on overcoming locality problems in first-order learning using relational paths was [Vere 1977], which introduced the idea of "association chains" composed of determinate binary relations. This idea foreshadows both determinate literals and *Relational Path Finding*.

Relational clichés is another method for dealing with locality problems [Silverstein & Pazzani 1991]. In this approach, the learning system has a predefined set of templates describing combinations of relations which often appear together.

The method of *Relational Path Finding* [Richards & Mooney 1992] is more general than *Relational clichés* since it does not depend on predefined templates; however, if the predefined templates are adequate for the learning domain, using *relational clichés* will be more efficient. This ap-

proach is different from our approach RPBL [Gao et al., 2008]. *Relational Path Finding* uses relational paths to improve FOIL algorithms to *escape* from local maxima and local plateaus. However, RPBL uses relational paths as explanations of positive examples, and *avoid* local maxima and local plateaus. In essence, RPBL is a combination of inductive learning with analytical learning, and *Relational Path Finding* belongs to inductive learning. RPBL generalizes relational paths to get hypothesis, instead of using relational paths as heuristics for adding literals. In addition, RPBL is a simultaneous covering approach, and *Relational Path Finding* is a sequential one.

Ong et al. extended *Relational Path Finding* and presented an algorithm named *Path Finding* in moded programs [Ong et al. 2005]. Their approach was based on the idea that the saturated clause can be represented as a hypergraph, and the use of hypergraphs to compose the final paths. They took advantage of mode information to reduce the number of possible paths and generate only legal combinations of literals. However, their approach generated longer clauses, which might be more vulnerable to *overfitting*.

RPBL uses a simultaneous covering strategy [Gao et al., 2008], and is based on searching relational paths in structured instance space. Our intuition is that if two nodes in structured instance space interact, there must exist an explanation of the interaction. We propose that the explanation should be a path linking the two nodes. However, we did not discuss the following problems in RPBL in the former paper [Gao et al., 2008]. The first problem is how to make use of domain theories, the second one is how to learn recursive theories, and the third one is the reason why RPBL performs better than standard ILP approaches. Therefore, we extend RPBL in these directions in this paper.

The rest of this paper is organized as below. We first outline the approach RPBL. Second, we extend it by learning with domain theories and learning recursive theories. We test our approach on several data sets, and experimental results are analyzed theoretically. At last, we conclude this paper.

Table 1. The outline of RPBL

RPBL (*Target_predicate, Precicates, Examples, Number, Length*)

- *Pos* <- those *Examples* for which the *Target_predicate* is *True*
- Construct *SIS* from *Pos*
- If domain theories exist, Update *SIS* to *E-SIS* by applying domain theories
- *Learned_rules* <- {}
- Select *Number* positive examples from *Pos* randomly
- For each *positive_example* selected, do
 - *RPs* <- Search-Relational-Paths (*positive_example, SIS, Length*)
 - *S-RPs* <- Specialize-Relational-Paths (*RPs, Examples*)
 - *G-S-RPs* <- Generalize-Specialized-Relational-Paths(*S-RPs*)
- Combine-Horn-Clauses(*G-S-RPs, Examples*)
- Return *Learned_rules*

2. Outline of RPBL

Our approach RPBL introduces five notations, which are given below.

Structured Instance Space (SIS) It is a labeled ordinary graph, in which nodes are binary facts, and edges are constant arguments shared between the two binary facts.

Expanded SIS (E-SIS) It is the expansion of SIS by applying domain theories to SIS.

Relational Path (RP) It is a path in SIS, linking the two constant arguments of positive training examples.

Specialized RP (S-RP) It is the specialization of RP by adding unary facts for the constant arguments of binary facts on RP.

Generalized S-RP (G-S-RP) It is the most general first-order Horn clause of S-RP, which is obtained by variablizing constant arguments of unary and binary facts on S-RP.

The outline of RPBL is shown in Table 1. To elaborate, imagine a subroutine *Search-Relational-Paths* that accepts a positive training example and a maximum length *Length* for searching RPs as input. It searches RPs in SIS, and outputs a set of RPs, which links the two constant arguments of the positive example. We use the subroutine *Speciaze-Relational-Paths* to specialize each RP by appending unary facts for all the constant arguments appears on the RP. For each S-RP, we then generalize it by the subroutine *Generalize-Specialized-Relational-Paths*. At last, we combine these G-S-RPs, i.e. first-order Horn clauses, to get hypothesis with the best performance.

3. Two Extensions to RPBL

We have two extensions to RPBL in this section, i.e. learning with domain theories and learning recursive theories.

3.1 Learning with Domain Theories

We use the task of learning *daughter_of* relationship to demonstrate how to tackle domain theories. There are 5 unary facts, which are

```
male(Tom),           male(Ian),
female(Ann),        female(Mary),
female(Eve)
```

There are 4 binary facts, which are

```
mother_of(Ann, Mary), mother_of(Ann, Tom),
faher_of(Tom, Eve),   faher_of(Tom, Ian)
```

There are 2 positive examples, which are

```
daughter_of(Mary, Ann), daughter_of(Eve, Tom)
```

There are 2 rules in domain theories, which are

```
parent_of(x, y) :- father_of(x, y)
parent_of(x, y) :- mother_of(x, y)
```

We use these domain theories to generate inferred facts, which are *parent_of(Ann, Mary)*, *parent_of(Ann, Tom)*, *parent_of(Tom, Eve)*, and *parent_of(Tom, Ian)*. The SIS is expanded by appending inferred facts, as shown in Fig. 1. Dash frames are inferred facts, and dash lines are added edges. We can use the E-SIS and the algorithm described above to learn *daughter_of* relationship.

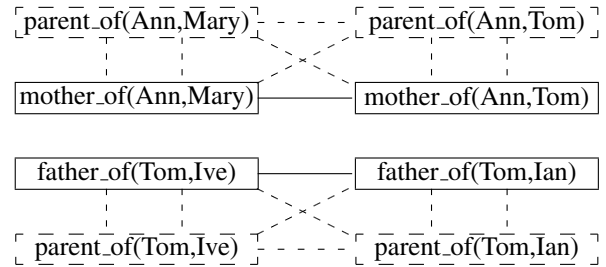


Fig. 1. Expanded structured instance space (E-SIS) for leaning *daughter_of*

3.2 Learning Recursive Theories

We use the task of learning *member* relationship on lists from a small world to demonstrate how to learn recursive theories. It contains the lists [], [1], [2], [3],

[1, 2], [2, 3], [1, 3], and [1, 2, 3]. The target relation $member(E, L)$ contains pairs whose first constant denotes an element that belongs to the list denoted by the second. In this small world there are just twelve tuples of $member$: $\langle 1, [1] \rangle$, $\langle 2, [2] \rangle$, $\langle 3, [3] \rangle$, $\langle 1, [1, 2] \rangle$, $\langle 2, [2, 3] \rangle$, $\langle 3, [2, 3] \rangle$, $\langle 1, [1, 3] \rangle$, $\langle 3, [1, 3] \rangle$, $\langle 1, [1, 2, 3] \rangle$, $\langle 2, [1, 2, 3] \rangle$, $\langle 3, [1, 2, 3] \rangle$. The first denoting that element 1 is a member of the list [1], and so on. Lists like [1,2,3] are just constants, so a background relation $components(L, H, T)$ is required to show how to find the head H and tail T of a list L. The tuples making up $components$ are: $\langle [1], 1, [] \rangle$, $\langle [2], 2, [] \rangle$, $\langle [3], 3, [] \rangle$, $\langle [1, 2], 1, [2] \rangle$, $\langle [2, 3], 2, [3] \rangle$, $\langle [1, 3], 1, [3] \rangle$, $\langle [1, 2, 3], 1, [2, 3] \rangle$. The first states that list [1] has head 1 and tail [], and so on.

We use $member(2, [1, 2])$ as an example. The RPs is:

```
member(2, [1, 2]) :- member(2, [2]),
    component([1, 2], 1, [2])
member(2, [1, 2]) :- component([1, 2], 1, [2])
```

Because there is no unary facts in this learning task, S-RP is the same as RP. So, its G-S-RPs or rules are

```
member(x, y) :- member(x, v1),
    component(y, v2, v1)      (rule 1)
member(x, y) :-
    component(y, v1, x)      (rule 2)
```

In rule 1 the predicate $member$ appears twice, both in the head and in the body. Therefore, we have learned its recursive definition. Part of the search tree is shown in Fig. 2.

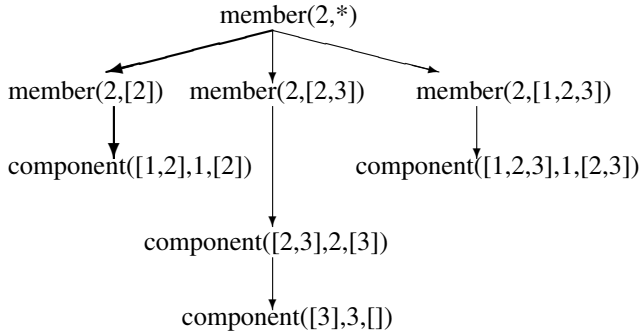


Fig. 2. Part of the search tree for learning recursive definition of $member$

4 Experiments

We conduct trials to see whether RPBL could learn recursive theories on $Member$. We then learn multiple target predicates on $Family$ data set. In order to test our approach on real data set, we run it to learn $Advised_by$ on UW-CSE

data set which is extracted from the web pages of Washington University. Because most of these data sets could not be divide into different folds, we use training set as test set except UW-CSE data set. Finally, we use assertions extracted from a real OWL DLP ontology to learn axioms, which is a benchmark ontology for ontology reasoning. Experimental results of RPBL are compared with standard ILP approaches such as FOIL, and state-of-the-art ILP approach such as Aleph.

Our algorithm RPBL is developed in Java on JDK 1.6, and FOIL program is run on the same machine. The machine have two 1.6G CPUs, and its main memory is 2G. The program of the FOIL program is downloaded from the web site of its inventor, R. Quinlan¹. The number of positive examples used to search relational paths, $Number$, is set to 4. The maximum length of paths, $Length$, ranges from 3 to 10 for different data sets. In the following tables, % for Precision, Recall, and F1-Measure is omitted. The unit of learning time, $Time$, is in *second*. When running FOIL program, we do not use negative examples, but accept Close World Assumption (CWA) as in RPBL. In addition, $Clauses$ is the number of clauses learned. $Length$ is the sum of lengths of all the clauses learned including the head literal.

4.1 Data set 1: Member

The data set for learning $Member$ is extracted from FOIL data sets, which could be downloaded from the personal web site of R. Quinlan. We use this data set to test the performance of learning recursive theories. In this experiment, we use all the examples as both training set and test set. The experimental result is given in Table 2. Experimental results on $Member$ show that F1-Measure of RPBL is better than FOIL. However, the description length of hypothesis learned by RPBL is longer than FOIL. The learning time of RPBL is shorter than FOIL.

4.2 Data set 2: Family

This data set $Family$ is extracted from the FORTE web-site². FORTE (First Order Revision of Theories from Examples) is a machine learning system for modifying a first-order Horn-clause domain theory (pure Prolog program) to fit a set of training examples. We use $family.dat$ to learn 12 family relationships. In this data set, there totally 744 positive examples and 1,488 negative examples. In this experiment, we use all the examples as both training set and test set. Experimental results on $Family$ data set in Table 3 show that the average F1-Measure of RPBL is better than FOIL. The average learning time of RPBL is shorter than FOIL. In

¹<http://www.rulequest.com/Personal/>

²<http://www.cs.utexas.edu/ftp/pub/mooney/forte>

some cases, the description length of RPBL is shorter than FOIL, however in other cases it is longer than FOIL.

4.3 Data set 3: UW-CSE

We use the UW-CSE data set for a study on a heavily relational data set, which is provided by [Richardson & Domingos 2004]. The data set concerns learning whether one entity is *Advised_by* other entity based on real data from the University of Washington CS Department. The example distribution is skewed as there are 113 positive examples versus 2,711 negative examples. The data set is particularly hard as each fold is very different from the other, thus performance on the training set may not carry to the testing set. Following the original authors, we divide the data into 5 folds, each one corresponding to a different group in the CS Department. For comparison, besides FOIL we also use the results of Aleph and *Path Finding*, see [Ong et al. 2005] for detail. As listed in Table 4 and 5, the average F1-Measure of RPBL is better than that of FOIL, Aleph, and *Path Finding*. The average description length of RPBL is shorter greatly than FOIL, Aleph and *Path Finding*. Additionally, the learning time of RPBL is shorter than FOIL. Table 5 is extracted from [Ong et al. 2005], and the learning time for Aleph and *Path Finding* is unknown.

4.4 Data set 4: OWL DLP Ontology

The OWL DLP ontology is downloaded from SEM-INTEC project³. We use the ontology package of Jena to create an *OntModel*, and use function *listIndividuals* in *OntModel* to extract all the individuals. We get totally 109,876 triples. There are a lot of object properties used in this ontology, however, most of them are not described in detail. We learn left object properties, which are listed in Table 6. Experimental results show that RPBL could successfully learn OWL DLP axioms from instances. Additionally, when we run FOIL on this data set, the main memory of our machine become full quickly, and no results could be obtained.

5. Analysis

In this section, we give an analysis for the reason why our approach RPBL performs better than standard ILP approaches, with respect to performance and complexity.

5.1 Performance

RPBL is a simultaneous covering approach. It learns multiple first-order Horn clauses at once. These clauses are

³<http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm>

then combined to get best performance. ILP approaches such as FOIL take the sequential covering strategy. They learn one rule once, removing the data it covers, then iterating this process. Therefore, RPBL searches a smaller space to get the best hypothesis, while FOIL searches a much larger space greedily, and often get sub-optimal hypothesis. In addition, RPBL uses relational paths as the explanation for positive training examples. So it needs less training examples compared to inductive learning approaches such as FOIL. It may also performs better when the number of training examples is not sufficient. At last, due to simultaneous covering strategy, RPBL does not remove the data it covers. Instead, its performance is measured on all the training examples. Additionally, *multiple* positive training examples are selected *randomly*. These strategies make it more robust than FOIL to noise of training examples.

5.2 Complexity

The time complexity of RPBL consists mainly of two parts, *searching relational paths* and *combining first-order clauses*. For the first part, we use a kind of breadth-first strategy, which requires the generation and storage of a tree whose size is exponential in the depth of the shallowest goal node. However, the number of expanded nodes will not exceed the number of constant arguments n . Because each constant argument is expanded at most once for each positive example. If there are m positive examples selected, then the complexity is $O(mn)$. For the second part, we use a depth-first strategy combined with a hill-climbing strategy. If there are b first-order clauses to be combined and the number of clauses in the best combination is limited to d , then the worst complexity is $O(b^d)$. Therefore, the worst complexity of our approach is exponential, which is $O(mn + b^d)$. Actually, the average complexity of RPBL is not so bad. Because we exclude those clauses with poor performances, b becomes smaller. Additionally, few clauses are needed to describe a target relation, thus d is small too. For example, on UW-CSE data set, one clause is learned to describe the target predicate, $d = 1$, and the complexity reduces to $O(mn + b)$.

6. Conclusions

RPBL is an relational paths based ILP approach, and experimental results show that it performs better than standard ILP approaches and some state-of-the art approaches. We use domain theories to expand structured instance space, and show that it can learn recursive theories by learning *member* relationship. We analyze its performance and time complexity and disadvantages. In addition, we give details of experimental results in this paper.

Table 2. Results of RPBL and FOIL on learning *Member*

Predicate	Algorithm	Precision	Recall	F1-Measure	Clauses	Length	Time
member	FOIL	40	100	57	2	3	0.1
	RPBL	77	91	83	2	5	0.019

Table 3. Results of RPBL and FOIL on *Family*

Predicate	Algorithm	Precision	Recall	F1-Measure	Clauses	Length	Time
brother	FOIL	30	96	45	5	14	0.1
	RPBL	95	96	96	2	6	0.031
uncle	FOIL	100	100	100	2	7	0.1
	RPBL	100	100	100	2	6	0.031
niece	FOIL	100	100	100	2	6	0.2
	RPBL	100	100	100	2	6	0.031
aunt	FOIL	100	100	100	3	9	0.4
	RPBL	100	100	100	2	6	0.031
nephew	FOIL	100	100	100	2	5	0.3
	RPBL	100	100	100	2	6	0.031
son	FOIL	100	100	100	2	6	0.4
	RPBL	100	100	100	2	6	0.031
mother	FOIL	100	100	100	1	2	0.4
	RPBL	100	100	100	2	6	0.031
father	FOIL	100	100	100	1	2	0.4
	RPBL	100	100	100	2	6	0.031
daughter	FOIL	100	100	100	3	7	0.5
	RPBL	100	100	100	2	6	0.032
sister	FOIL	100	66	79	1	7	0.7
	RPBL	100	98	99	2	6	0.015
wife	FOIL	100	100	100	1	1	0.7
	RPBL	100	100	100	1	3	0.015
husband	FOIL	100	100	100	1	1	0.7
	RPBL	100	100	100	1	3	0.016
Average	FOIL	94.2	96.8	93.7	2	5.58	0.4
	RPBL	99.6	99.5	99.6	1.83	5.50	0.027

Table 4. Results of RPBL and FOIL on learning *Advised by*

Fold	Algorithm	Precision	Recall	F1-Measure	Clauses	Length	Time
Theory	FOIL	100	6	12	2	14	190.0
	RPBL	25	57	35	1	5	0.41
AI	FOIL	75	9	15	2	10	894.3
	RPBL	38	40	39	1	5	2.56
Graphics	FOIL	0	0	0	2	16	727.7
	RPBL	50	45	47	1	5	4.39
Languages	FOIL	100	11	20	2	15	943.1
	RPBL	100	11	20	1	5	4.53
Systems	FOIL	37	39	38	3	19	624.3
	RPBL	37	39	38	1	5	2.33
Average	FOIL	62.4	13.0	17	2.2	14.8	675.9
	RPBL	50.0	38.4	35.8	1	5	2.84

Table 5. Results of Aleph and Path Finding on learning Advised_by

Fold	Algorithm	Precision	Recall	F1-Measure	Clauses	Length	Time
Theory	Aleph	27	38	32	2	9	-
	Path Finding	11	81	19	2	10	-
AI	Aleph	9	63	16	3	11	-
	Path Finding	12	75	21	1	11	-
Graphics	Aleph	46	85	60	3	12	-
	Path Finding	20	95	33	2	12	-
Languages	Aleph	100	22	36	1	3	-
	Path Finding	100	33	50	3	21	-
Systems	Aleph	8	87	15	1	4	-
	Path Finding	9	82	16	3	17	-
Average	Aleph	38.0	59	31.8	2.0	7.8	-
	Path Finding	30.4	73.2	27.8	2.2	14.2	-

Table 6. Results of RPBL on OWL DLP ontology

Predicate	Algorithm	Precision	Recall	F-Measure	Clauses	Length	Time
IsLoanOf	RPBL	100	89	94	2	16	30.4
hasOwner	RPBL	100	100	100	2	14	237.34
isCreditCardOf	RPBL	100	74	85	2	16	39.41
hasLoan	RPBL	100	89	94	2	16	24.66
isOwnerOf	RPBL	100	100	100	2	14	210.42
hasCreditCard	RPBL	100	82	90	3	24	36.15
Average	RPBL	100	89	93.8	2.17	16.67	96.40

7. Acknowledgments

This work is supported by National Science Foundation of China under Grant 60773107, 60873153, 60803061 and National Key Basic Research and Development Program of China under Grant 2003CB317004.

References

- [Gao et al., 2008] Gao Z. Q., Zhang Z. Z., and Huang Z. S.: Learning Relations by Path Finding and Simultaneous Covering. Submitted to World Congress on Computer Science and Information Engineering (CSIE 2009).
- [Ong et al. 2005] Ong I. M., Dutra I. C., Page D., Costa V. S. 2005. Mode Directed Path Finding. *Proceedings of 16th European Conference On Machine Learning*, 673-681.
- [Pazzani et al. 1991] Pazzani M. J., Brunk C. A., Silverstein G. 1991. A Knowledge-intensive Approach to Relational Concept Learning. *Proceedings of the Eighth International Workshop on Machine Learning*, 432-436.
- [Quillian 1990] Quinlan J. R. 1990. Learning Logical Definitions from Relations. *Machine Learning*, 5:239-266.
- [Richards & Mooney 1991] Richards B. L., Mooney R. J. 1991. First-Order Theory Revision. In *Proceedings of the Eighth International Workshop on Machine Learning*, 447-451.
- [Richards & Mooney 1992] Richards B. L., Mooney R. J. 1992. Learning Relations by Pathfinding. In *Proceedings of the Tenth Conference on Artificial Intelligence*, 50-55.
- [Richardson & Domingos 2004] Richardson M., Domingos P. 2004. Markov Logic Networks. *Technical Report*, Dept. of Computer Science and Engineering, University of Washington, Seattle, WA.
- [Silverstein & Pazzani 1991] Silverstein G., Pazzani M. J. 1991. Relational Cliches: Constraining Constructive Induction During Relational learning. In *Proceedings of the Eighth International Workshop on Machine Learning*, 203-207.
- [Vere 1977] Vere S. A. 1977. Induction of Relational Productions in the Presence of Background Information. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 349-355.