# An ILP Method Based on Instance Graph

**Runqi Zhang**
Department of Computer Science and Engineering
State University of New York at Buffalo
rzhang@cse.buffalo.edu

A necessary function of ILP systems is to test whether rulesets intensionally cover examples. In general, when R contains recursive clauses, the above test may not terminate. In order that ILP techniques can serve end users directly in applications such as knowledge discovery in databases, general methods must be provided to ensure that the learned rulesets are executable and, in particular, that they do not lead to infinite recursion. To the best of our knowledge, all current methods are based on some kind of "Literal Order" or analog. Among them, the method "Ordering a Set of Constants and then Ordering Recursive Literals" that FOIL used (Cameron-Jones & Quinlan 1993) is one of the most advanced. However, this method suffers in at least three ways. ❶It can not be extended to Multiple Predicate Learning (MPL) (Raedt et cl. 1993) that involves more than one relation, e.g. when relation R invokes S and S invokes R. ❷It may not order constants correctly so as to have to rely on users to define constant orders for many recursive learning tasks. Sometimes this is necessary but not easy for users. ❸It needs to define theory constants that are strong hints of ground clauses.

In our opinion, in order to overcome the above shortcomings, FOIL should memory more of previous learning to help further search in rule space. If we can efficiently record and update the effects that are made to the instance space every time when a new (recursive) clause is added to the rule, we can design more powerful heuristics to help search the rule space. Based on this idea, we invented **instance graph $H_{(R,E)}$**, where R is a ruleset and E is an instance space.

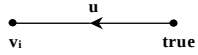


Fig. 1

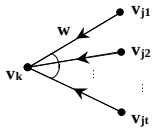


Fig. 2

Instance graph is a type of directed hypergraph. In instance graph, every vertex denotes an example of the target predicate. Every hyperarc in instance graph $H_{(R,E)}$ denotes an instance of a clause in R that contains no negative background examples in its body. There are two types of hyperarcs. The first type illustrated in Fig. 1 denotes an instance of a non-recursive clause. $v_i$ denotes the head of the instance, and "true" is a special vertex we defined. The second type illustrated in Fig. 2 denotes an instance of a recursive clause. $v_k$ denotes the head of the instance and $v_{j1},...,v_{jt}$ denote all target examples appearing in the body of the instance. Clearly, instance graph $H_{(R,E)}$ can detail the relationship between ruleset R and target example space E. An example is **intensionally covered** by R if and only if there is a hyper-path from "true" to the vertex that denotes the example in $H_{(R,E)}$.

Because of the powerful description of instance graph, we can take account of **hung** examples rather than just covered and abandoned examples (a hung example is an example that is extensionally covered by current ruleset and may be intensionally covered after some new clause(s) added into the ruleset).

We designed a data structure to maintain instance graph efficiently. Then we designed a new ILP algorithm FOILBIG (First-Order Inductive Learner Based on Instance Graph). FOILBIG guarantees that i**f FOILBIG believes a ruleset R covers an example e, then there certainly is a hyper-path from "true" to the vertex that denotes e in the instance graph, i.e. R intensionally covers e**. FOILBIG guarantees executability of learned rulesets, and does not substantially raise computational complexity compared to FOIL. FOILBIG has no restriction of ordering, and hence makes it possible to complete more learning tasks in ILP context. The method based on instance graph could also be used by any learning system that grows elements from ground facts by repeated specialization. FOILBIG has been implemented superficially and preliminary experiment shows that it can solve problems beyond the scope of FOIL.

In addition, unlike the method used by FOIL, our method can be extended to MPL with the extension of instance graph. We believe this is truly significant because MPL is much more complex than SPL (Single Predicate Learning) (detailed in Raedt et cl. 1993).

Part of our research has been published (Zhang et cl. 1999). See http://www.cse.buffalo.edu/~rzhang/research for more details.

**Acknowledgments** Thanks to my advisors Prof. Xiaoping Chen and Prof. Xin He, and colleague Guiquan Liu. Special thanks to Prof. Robert L Givan for his comments and suggestions, and to my friend Perry Caldwell for his constant help.

## References

Cameron-Jones, R.M. and Quinlan, J.R. 1993. Avoiding pitfalls when learning recursive theories. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*. 1050-1055.

Raedt, L.De; Lavrac, N. and Dzeroski, S. 1993. Multiple predicate learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.* 1037-1042.

Zhang, R., Chen, X. and Liu, G. 1999. An ILP algorithm without restriction of constant ordering. *The Chinese Journal of Software* (10)8: 868-876.