On Applying Tabling to Inductive Logic Programming

Ricardo Rocha – Nuno Fonseca – Vítor Santos Costa
LIACC & Univ. Porto – LIACC & Univ. Porto – Univ. Wisconsin, Madison

Introduction

It is recognized that efficiency and scalability is a major obstacle to an increased usage of Inductive Logic Programming (ILP) in complex applications with large hypotheses spaces. In this work, we focus on improving the efficiency and scalability of ILP systems by exploring tabling mechanisms available in the underlying Logic Programming systems. We present two different approaches. Our first approach is a direct application of tabling to query execution. The second approach is designed to take advantage of the redundancy in ILP search. To validate our approaches, we ran the April ILP system in the YapTab Prolog tabling system using two well-known datasets. The results obtained show quite impressive gains without changing the theories generated.

Tabling

Tabling is an implementation technique where results for subcomputations are stored and then reused when a repeated computation appears.

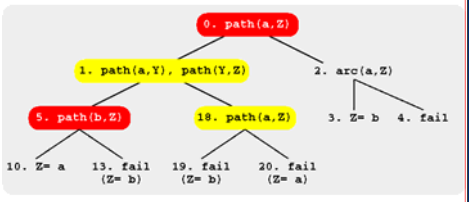
The basic idea behind tabling is straightforward:

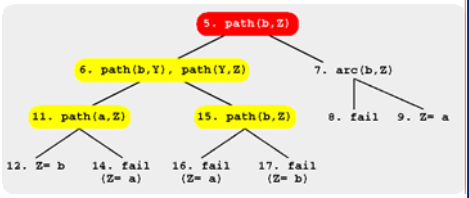
- Programs are evaluated by storing newly found answers of current subgoals in an appropriate data space, called the table space.
- New calls to a predicate check this table to verify whether they are repeated. If they are, answers are recalled from the table instead of the call being re-evaluated against the program clauses.
- Meanwhile, as new answers are found, they are inserted into the table and returned to all variant subgoals.

```
:- table path/2.

path(X,Z):- path(X,Y),
            path(Y,Z).
path(X,Z):- arc(X,Z).

arc(a,b).
arc(b,a).

?- path(a,Z).
```

Table Space

Subgoal	Answers
0. path(a,Z)	3. Z= b 10. Z= a
5. path(b,Z)	9. Z= a 12. Z= b





Tabling has proven to be particularly effective in Prolog programs:

- **Avoids recomputation**
- **Avoids infinite loops**

Tabling has been successfully applied to real applications:

- Model Checking
- Program Analysis
- Deductive Databases
- Non-Monotonic Reasoning

ILP

A general ILP system spends most of its time evaluating hypotheses, either because the number of examples is large or because testing each example is computationally hard.

An important characteristic of ILP systems is that they generate candidate hypotheses (clauses) which have many similarities among them.

- hyp(X):- a(X).
- hyp(X):- a(X), b(Y).
- hyp(X):- a(X), b(Y), c(X,Y).
- hyp(X):- a(X), b(Y), d(Y,Z).

Computing the coverage of an hypothesis requires, in general, running all positives and negatives examples against the clause. For example, to evaluate if the example hyp(e1) is covered by the previous hypotheses, the system executes the goals:

- a(e1).
- a(e1), b(Y).
- a(e1), b(Y), c(e1,Y).
- a(e1), b(Y), d(Y,Z).

We can do a lot of recomputation!

- a(e1) is executed 4 times.
- b(Y) is executed 3 times.
- a(e1), b(Y) is executed 3 times.

Tabling for ILP

This suggests two approaches to avoid recomputation:

- We can **table subgoals**. This requires no changes to the ILP system. We simply need the declare the predicates to table. This approach will only work if the computation for a subgoal is expensive. It will bring no benefit if, say, the subgoal reduces to a database access.
- We can **table conjunction of subgoals**. This approach is only useful if we repeatedly generate the same prefix. If we have a large number of prefixes which are only called a few times, we may need large amounts of space to store the tables, and gain little time-wise.

Tabling conjunction of subgoals requires minimal changes to the ILP system. We designed the following solution. All clauses defined by a conjunction of N subgoals (N>2) are redefined to use two literals. The first defines the conjunction of the N-1 initial subgoals. The second is the N subgoal.

- hyp(X):- a(X), b(Y), c(X,Y).
  → hyp(X):- a_b(X,Y), c(X,Y).
- hyp(X):- a(X), b(Y), c(X,Y), d(Z).
  → hyp(X):- a_b_c(X,Y,X,Y), d(Z).
- hyp(X):- a(X), b(Y), c(Z,Z), d(Z).
  → hyp(X):- a_b_c(X,Y,Z,Z), d(Z).

The conjunctions are defined in tabled predicates. As different conjunctions are generated, the system dynamically asserts a new tabled predicate that abstracts the set of arguments in the conjunction.

- :- table a_b/2.
  a_b(X1,X2):- a(X1), b(X2).
- :- table a_b_c/4.
  a_b_c(X1,X2,X3,X4):- a_b(X1,X2), c(X3,X4).

Note that, when calling these predicates, this may cause the same variables to appear at several positions. In practice, this is not a problem because the tabling engine only stores the answers once for each different variable.

This idea can be recursively applied as the system generates more specific hypothesis. This idea is similar to the query packs technique.

Initial Results: April + YapTab

To evaluate the impact of using tabling in real application problems, we ran the April ILP system with the YapTab Prolog tabling system using two ILP datasets. The environment for our experiments was an AMD Athlon MP 2600+ processor with 2 GBytes of main memory and running the Linux kernel 2.6.11.

To evaluate hypotheses we experimented with three different approaches: no tabling; subgoals being evaluated using tabling; and subgoals and conjunction of subgoals being evaluated using tabling.

	Running time (s)			Table usage (Mb)	
# hyps	no tab	tab sub	tab conj	tab sub	tab conj
981	> 4 hours	94	92	2	6
6,514	na	162	140	5	205
14,020	na	169	146	6	281
20,299	na	197	mo	6	mo
26,484	na	219	mo	6	mo
32,852	na	236	mo	6	mo

Mutagenesis dataset

	Running time (s)			Table usage (Mb)	
# hyps	no tab	tab sub	tab conj	tab sub	tab conj
998	1	1	1	3	11
9,998	7	9	13	11	259
19,998	81	91	mo	11	mo
29,932	121	124	mo	11	mo
39,932	161	154	mo	11	mo
49,869	225	209	mo	12	mo

Carcinogenesis dataset

Further Work

A major problem when tabling conjunction of subgoals is that we can increase table memory usage arbitrarily.

- A simple solution is to abolish the full set of tables from the table space when we run out of memory.
- An alternative would be to abolish the tables potentially useless when we backtrack in the hypotheses space. This later approach requires further study to avoid incorrect deletions.
- Another alternative would be to store all the answers in an unique table shared by all tabled predicates.

Conclusions

We propose the ability of using tabling mechanisms available in the underlying Logic Programming systems to minimize recomputation in ILP systems.

- First, tabling can be used to reduce the search space by avoiding recomputation for the subgoals evaluated during the coverage of the hypotheses.
- Second, by tabling the conjunction of subgoals in the hypotheses we can further reduce the search space and improve performance.
- Third, because tabling based models are also able to avoid infinite loops, they can ensure termination for a wider class of programs. The latter can be useful when dealing with datasets with recursive definitions in the background knowledge.

Acknowledgements

This work has been partially supported by APRIL (POSI /SRI/40749/2001), Myddas (POSC/EIA/59154/2004), US Air Force (F30602-01-2-0571), and by funds granted to LIACC through the Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia (FCT) and Programa POSC. Nuno Fonseca is funded by the FCT grant SFRH/BD/7045/2001.