Phase Transition and Heuristic Search in Relational Learning

Erick ALPHONSE and Aomar OSMANI
LIPN-CNRS UMR 7030, Université Paris 13, France

Abstract

Several works have shown that the covering test in relational learning exhibits a phase transition in its covering probability. It is argued that this phase transition dooms every learning algorithm to fail to identify a target concept lying close to it. However, in this paper we exhibit a counter-example which shows that this conclusion must be qualified in the general case. Mostly building on the work of Winston on near-misse examples, we show that, on the same set of problems, a top-down data-driven strategy can cross any plateau if near-misses are supplied in the training set, whereas they do not change the plateau profile and do not guide a generate-and-test strategy. We conclude that the location of the target concept with respect to the phase transition alone is not a reliable indication of the learning problem difficulty as previously thought.

1 Introduction

It was discovered early on that learning in relational languages, also known as Inductive Logic Programming (ILP), had to face important *plateau phenomena*: the evaluation function, used to prioritize nodes in the refinement graph is constant in parts of the search space, and the search goes blind. These plateau phenomena are the pathological case of heuristic search, complete or not [11]. An explanation can be given after the work of [4], who studied the ILP covering test within the phase transition framework. As illustrated in figure 1, the covering test is NP-complete and therefore exhibits a sharp PT in its coverage probability [5]. When one studies the probability of covering a random example of a fixed size by a hypothesis given the hypothesis' size, one distinguishes three well-identified regions: a region named "yes", where the probability of covering an example is close to 1, a region named "no", where the probability is close to 0, and finally in between the phase transition, named the "mushy" or the "pt" region, where an example may or may not be covered. As the heuristic value of a hypothesis depends on the number of covered examples (positive or negative), we see that the two regions "yes" and "no" represent plateaus that need to be crossed during search without an informative heuristic value.

To investigate the impact of the occurrence of a PT in the covering test on the learning success rate, systematic experiments with several learning algorithms have been conducted on a large set of artificially generated problems by [4]. The

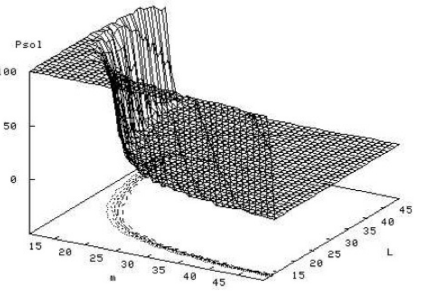


Figure 1. From [4]: Coverage probability P_{sol} of an example with L constants by a hypothesis with m literals. The contour level plots, projected onto the plane (m, L), correspond to the region where $P_{sol} \in [0.01, 0.99]$.

authors generated a set of 451 problems by choosing each target concept according to its location in the (m, L) plane with respect to the PT, as shown in figure 1. Figure 2 summarizes the application of FOIL on this set of problems: learning failure or success is noted for each problem identified by the (m, L) pair. One important conclusion of their work is that the occurrence of a PT in the covering test is a general problem for all learning algorithms: the PT is viewed as an "attractor" for the heuristic search of any learning algorithm, which is bound to find a concept definition in the PT.

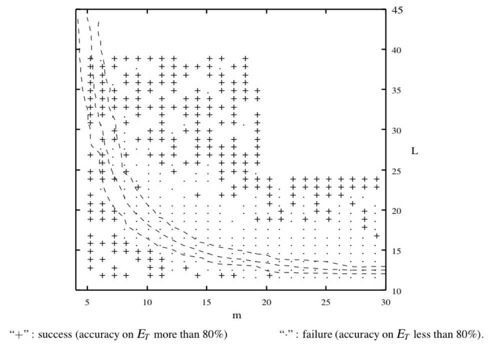


Figure 2. From [4]: FOIL's competence map: success and failure regions. The PT region is indicated by the dashed curves, corresponding to Psol = 0.9, Psol = 0.5 and Psol = 0.1

We note however that only generate-and-test (GT) learning algorithms have been investigated in this work and that this conclusion has to be qualified in the general case. In the GT paradigm, refinements are only based on the structure of the hypothesis space, independently of the learning data. Therefore, for a given hypothesis, GT algorithms have to deal with many refinements that are not relevant with respect to the discrimination task. On the contrary, data-driven strategies (DD) allow to use the training data to prune irrelevant branches of the refinement graph before relying on the evaluation function and may overcome the problem of plateaus. Notably, building on the pioneering work of Winston on near-miss examples [16], we show, as a counter-example, that, on the same set of problems as [4], a Top-down Data-Driven learner (TDD) can cross any plateau and reach the target concept whenever near-misses are supplied in the training set, whereas these near-misses do not change the plateau profile and do not guide a TGT learner. This counter-example exhibits a class of problems where the hypothesis search is trivial although the covering test acts as a plateau. We conclude that the location of the target concept with respect to the PT *alone* is not a reliable indication of the learning problem difficulty, as previously thought.

In the next section, we recall the two top-down learning strategies that we investigate in this paper, namely the TGT and the TDD strategies. In section 3, we draw a parallel between the PT of the covering test and the evaluation function of a learning algorithm. Next, in section 4, we show how we can add near-misses to those problems in such a way that the plateaus are unchanged in order to guide a TDD algorithm to the target concept without search in the hypothesis space. This shows in particular that plateaus are pathological for TGT algorithms but are not a complexity criterion for all algorithms.

2 Top-down generate-and-test and data-driven strategies

In concept learning, we are given a learning set $E = E^+ \cup E^-$, with positive and negative examples of the unknown target concept, drawn from an example or instance language $\mathcal{L}_e$, a hypothesis language $\mathcal{L}_h$, a generality relation named covering test $\geq$ which relates $\mathcal{L}_e$ and $\mathcal{L}_h$ and partially order the hypotheses. After [10], concept learning is defined as search in $\mathcal{L}_h$. The problem, known as the consistency problem, is to find a hypothesis $h \in \mathcal{L}_h$ such that h is consistent with the data: $\forall e^+ \in E^+, h \geq e^+$ *(completeness)* and $\forall e^- \in E^-, h \not\geq e^-$ *(correctness).*

This seminal paper, relating (symbolic) concept learning to search in a state space, has enabled machine learning to integrate techniques from problem solving, operational research and combinatorics: greedy search in C4.5 and FOIL beam search in AQ and CN2, breadth-first search and Aleph, 'A' search in PROGOL, IDA (Iterative-Deepening A) search in MIO to name a few systems. However, search in languages of interest, as those typically considered in ILP, is NP-hard (e.g. [7, 8]) and heuristic search is crucial for efficiency [11]. Heuristic search makes use of an evaluation function to prioritize nodes to consider at each stage of the search. A survey of evaluation functions used in machine learning can be found in [6]. It shows that all of them are based, without loss of generality, on three parameters that are the coverage rate of positive examples, the coverage rate of negative examples and a complexity measure of the hypothesis under consideration. The first two parameters are inherited from the learning task definition given above and stress the impact of the relevance of the covering test on the quality of search.

A central idea in concept learning is the use of a generality partial order between hypotheses to guide the resolution of the consistency problem [10]: if a hypothesis is incorrect, it is said to be too general and has to be specialised. The natural stopping criterion[1] of a top-down strategy is the acquisition of a correct hypothesis. In the rest of the article, we will consider that learning of a clausal theory is done with the covering strategy. Mitchell further refines the search strategy into the generate-and-test and data-driven strategies.
In the GT paradigm, the top-down refinement operator, noted ρ, is only based on the structure of the hypothesis space, independently of the learning data:

$$\text{Let } h \in \mathcal{L}_h : \rho(h) = \{h' \in \mathcal{L}_h | h \geq h'\}$$

Therefore, generate-and-test algorithms have to deal with many refinements that are not relevant with respect to the discrimination task. They only rely on the evaluation function to prune the irrelevant branches. On the contrary, the DD strategy searches the space of hypotheses that are more general than or equal to a given positive example, named the seed example, and uses negative examples to prune irrelevant branches in the refinement graph. Formally, the TDD refinement operator is defined as a binary operator:

$$\text{Let } h \in \mathcal{L}_h, e^- \in E^- : \rho(h, e^-) = \{h' \in \mathcal{L}_h | h \geq h' \, and \, h' \not\geq e^-\}$$

As opposed to a TGT approach, a TDD approach can therefore compensate for a poor evaluation function by using the learning data. Relying on the negative examples allows a TDD strategy to have a branching factor that is smaller than the branching factor of a generate-and-test strategy searching in the same hypothesis space. Moreover, some TDD strategies make the most of the negative instances in order to select informative negative examples only [16, 14, 2], e.g. *near-misses* after Winston. A near-miss in the Winston's

[1]In the article, only the noise-free learning setting is considered. In the case of noise, this stopping criterion is relaxed to accept hypotheses that are not consistent.

terminology is a negative example that differs from the seed example by only one description (a literal or an attribute of the positive example that is not in the negative example). This description necessarily belongs to the hypothesis that discriminates them. Ultimately, a TDD algorithm learning from a dataset provided with all near-misses with respect to the target concept would converge to the concept without search, generating only one refinement at each step. This is the result we are going to use in ILP to overcome the problem of plateaus exemplified in the PT framework [4], showing that plateaus do not necessarily imply a high complexity of search in learning.

3 Covering test PT and learning success rate

This section investigates the impact of the PT of the covering test on the quality of the evaluation function and subsequently on the efficiency of the heuristic search.We re-use the set of learning problems presented in [4] but in a different setting where there is a direct parallel between the plateaus of the "yes" and "no" regions in figure 1 and the plateau of the heuristic search. As shown in [4], the hypothesis space where FOIL is run is different from the hypothesis space where the PT is drawn. Precisely, the learning algorithms are allowed to add several literals based on the same predicate symbol and they evaluate hypotheses whose coverage rates are different from those pictured in the figure of the PT (figure 1). We conduct the same experiments but FOIL is bound to search in the same hypothesis space as the one where the PT is exhibited. To run the experiments, we use Aleph [15] to emulate FOIL, as FOIL, as far as we know, cannot be set to run in such a hypothesis space.

3.1 Problem description

Botta et al. tackle the learning of a single Datalog clause from the hypothesis space $\mathcal{L}_h^m$ built as follow:

$$\mathcal{L}_h^m = \{c \leftarrow \bigwedge_{k=1}^{n-1} p_{l_k}(X_k, X_{k+1}) \wedge \bigwedge_{k=n}^{m} p_{l_k}(X_{i_k}, X_{j_k})\}$$

where c is the clause head without variables, $i_k < j_k \in \{1, \ldots, n\}, l_k \in \{1, \ldots, m\}$ and such that all literals in the clause body are built on distinct binary predicate symbols. Examples are represented as ground Datalog clauses. Formally, let $A = \{a_1, \ldots, a_L\}$ be the set of all constants of cardinality L, and N the number of literals per predicate symbol. The example language $\mathcal{L}_e$ is the set of ground Datalog clauses defined as follow:

$$\mathcal{L}_e = \{c \leftarrow \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{N} p_i(a_{ij^1}, a_{ij^2})\}$$

Each pair of constants (a_{ij^1}, a_{ij^2}) is drawn uniformly without replacement from the set $A \times A$.

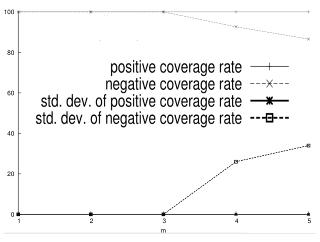
A learning problem is parametrized with the tuple (m, L). The number of variables n is fixed to 4 and the number of literals per predicate symbols N is fixed to 100. In a (m, L) problem, m is the size of the target concept drawn from $\mathcal{L}_h^m$ and L the number of constants in $\mathcal{L}_e$. For each problem, a learning set and a test set are built. Both are balanced sets of examples with 100 positive examples and 100 negative examples. In order to visit as uniformly as possible the "yes", "no" and "pt" regions, (m, L) problems have been uniformly selected without replacement for $m \in [5, 29]$ and $L \in [12, 39]$. It has to be noted that if (m, L) lies in the "yes" (resp. "no") region, by construction the concept description will almost surely cover (resp. reject) any randomly constructed example. For those problems, The example generator is modified and relies on a repair mechanism to ensure a balanced distribution of positive and negative examples [4].

3.2 Facing the plateau with a TGT strategy

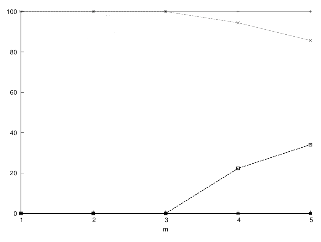
We ran FOIL on problems on m=5 and 10 lines and on the upper-right corner problems ranging from $L \in [24, 39]$ on the m=18 line and from $m \in [18, 29]$ on the L=24 line. We also sampled some other problems without any difference in the results. On all of these problems, FOIL is unable to find any good approximation of the target concepts, being in the "yes", "no" or "pt" regions. Note that this result differs from [4] where FOIL, in a different hypothesis space, was shown to fail only in the "pt" region and in the "no" region for small numbers of constants L (see figure 2). To show why, we plot for a problem the coverage rate of the positive and the negative examples, as well as their standard deviation depending on the size of the hypothesis, averaged over 1000 randomly and uniformly drawn hypotheses as in [4]. A plateau is materialised by a standard deviation of the coverage rates of the examples close to 0.

The left side of figure 3 shows the results for the (5,15) problems in the "yes" region, (10,20) in the "pt" region and (14,28) in the "no" region. For each problem, the value of a hypothesis, given an evaluation function, can be read on average from the corresponding sub-figure. As the top-down learner searches the hypothesis space, it evaluates hypotheses in turn of increasing size. For instance, on the $(5, 15)$ problem, we can see that all positive examples are covered (100% of positive coverage rate) for any hypothesis size, and all negative examples up to a size of 3. Whatever the evaluation function is, the top-down learner will see hypotheses up to 3 literals long of equal value, or equivalently, it has to cross a plateau of width 3 before being able to use the evaluation function to discriminate between hypotheses.

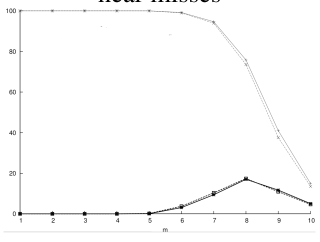
As we can see for the "yes" and the "pt" regions, hypotheses at the bottom of the search space are incorrect (i.e. for the maximum hypothesis size): on average, there does



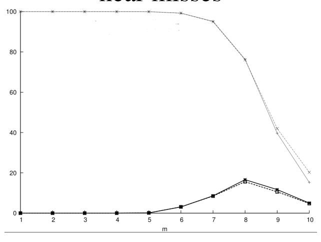
(5,15): ”yes” region without near misses



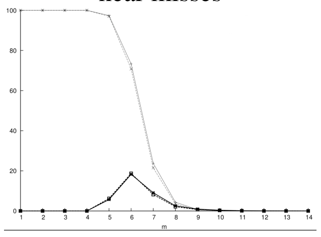
(5,15): ”yes” region with near misses



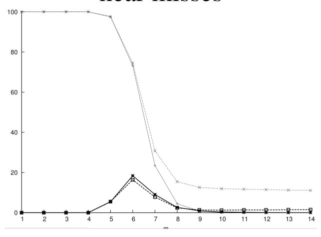
(10,20): ”pt” region without near misses



(10,20): ”pt” region with near misses



(14,28): ”no” region without near misses



(14,28’): ”no” region with near misses

Figure 3. Coverage rates and plateau profiles for representative “pt” problems in the “yes”, “pt” and “no” regions.

not exist a correct theory of the learning problems in these regions. The target concept is hidden among the bottom hypotheses and a TGT algorithm can only find it by chance as we will detail later. The natural stopping criterion of the TGT strategy is only met for problems generated in the “no” region and a TGT learner is bound to output hypotheses belonging to the intermediate region in between the “pt” and the “no” regions. This is similar to the finding of [4]. Therefore, we ran FOIL on several problems in the “no” region, whose results of typical cases are summarised in table 1.

Table 1. Accuracy measured on the test set.

(m,L)	(8,34)	(14,17)	(14,38)	(19,35)	(29,24)
Accuracy (%)	54.5	47.5	54.5	58	46
# of clauses	91	55	66	54	38
$\langle min, max\rangle$	$\langle 7,7\rangle$	$\langle 7,8\rangle$	$\langle 6,8\rangle$	$\langle 13,14\rangle$	$\langle 8,9\rangle$

This table gives, for each problem, the accuracy evaluated on the corresponding test set, the number of clauses in the theory and the minimum and maximum size of the clauses in the theory. Again FOIL is unable to produce any reasonable approximations of the target concept and performs seemingly as a random classifier. These results are not surprising as we can see that the “yes” region of the phase transition represents plateaus for the heuristic search that must be crossed without an informative evaluation function to guide the search toward the good path to the target concept.

The impact of plateaus is well-known on classical heuristic search and they are a pathological case of heuristic search, whether complete or not, as the search goes blind (see e.g. [11, 9, 13]). For instance, in the case of a greedy search such as used in FOIL, the algorithm has to make a random choice between possible refinements and therefore acts as a random walker. In one of the smaller problems $(5, 15)$, the plateau is already as wide as 3 literals. FOIL’s search can be seen as starting the top-down search from a randomly and uniformly drawn partial hypothesis of size 3. It appears on all learning problems that a plateau of such width, which lengthens as we take problems with smaller number of constants L, is enough to fool the search. Several approaches have been proposed to overcome the limitation of a greedy search in ILP like look-ahead [3] or macro-operators [1] but we do not discuss these techniques here as their effectiveness strongly depends on the width of the plateau and we rather focus on complete heuristic searches. This is also a pathology for complete heuristic searches, like the best-first or the best-first iterative-deepening searches [9] implemented in PROGOL and MIO respectively, although they can theoretically find the target concept in any of the considered learning problems (see below). Best-first search (and its relatives A or A*) cannot be guided by the evaluation function and it therefore degenerates into a breadth-first search where the space requirement grows exponentially. It is often said that A-like search runs out of memory before reaching the time limit [9]. Best-first iterative-deepening (and its relatives IDA or IDA*), used in MIO, avoid the exponential space complexity but will reach the time limit.

The state of the art on evaluation functions used in learning shows that they are all, without loss of generality, based on two main parameters that are the rate of positive and negative examples covered. As these two parameters are inherited from the definition of the learning task, it is unlikely that a solution to overcome these plateau phenomena consists in designing new evaluation functions. The TGT learning paradigm seems doomed to fail to identify a good approximation of the target concept when facing non-trivial plateaus.

4 Crossing the plateau using near-misses with a TDD strategy

In this section, we show that near-misses can be added to a learning problem without changing the plateau profile.

In doing so, a TGT learner cannot take advantage of the addition of the most informative negative examples whereas a TDD learner is guided to the target concept without search and then overcomes the plateau problems. For the sake of readability, we quickly present the TDD strategy in ILP, limited to our setting, and we refer to [2] for a detailed presentation.

4.1 The TDD strategy in ILP

The TDD strategy is biased toward the covering of a seed example, s, and then amounts to map the initial search space of the learning algorithm into the space of generalisations of the seed example. Additionally, it is required that this space of generalisations be isomorphic to a boolean lattice and therefore the TDD refinement operator is an algebraic resolution of the learning problem. Looking for a hypothesis of $\mathcal{L}_h^m$ that both covers s and rejects a negative example e^- can be equivalently performed by looking for a generalisation of s that rejects $lgg(s, e^-)$, the least general generalisation of s and e^-. By definition of the *lgg* [12], we have $h \geq s \wedge h \geq e^- \Leftrightarrow h \geq lgg(s, e^-)$. Equivalently, by transposition, we have $h \ngeq s \vee h \ngeq e^- \Leftrightarrow h \ngeq lgg(s, e^-)$. As the TDD strategy is biased toward the covering of s, $h \ngeq e^- \Leftrightarrow h \ngeq lgg(s, e^-)$. Note that in ILP the lggs are not unique and several lggs need to be rejected before rejecting the initial negative example.

By reformulating the whole negative example set into a boolean search space by means of lggs, we can show that only the most specific set of lggs, which are incomparable by definition, are useful for learning. By making use of this partial order between negative examples, the TDD strategy can reject the most informative negative examples only and following that way efficiently prune the refinement graph. Precisely, if a negative example is such that one of its lggs with the seed misses only one literal of the seed, then this literal necessarily belongs to the hypothesis that discriminate them. This particular example, or its particular lgg, is a Winston's near-miss. The TDD strategy implemented in PROPAL makes the most of this property by rejecting the closest examples first, named in this context the nearest-misses examples. A Winston's near-miss is a nearest-miss by definition, being the closest.

4.2 Generating near-misses in the artificially generated problems

To guide the search with near-misses, we have to create new negative examples which differ from the seed example only by one literal in the reformulated space by means of lggs. We will have as many near-misses as literals in the target concept to guide the refinement process literal by literal. Before presenting the generation procedure, we have to set the seed example, or equivalently the bottom clause of the search space. For efficiency reasons, and without loss of generality, we do not choose a positive example as seed. The reason is that the examples provided in the set of problems can be very large with up to 3000 literals. Instead, we choose as a bottom clause the union of the most specific clauses in the hypothesis space, taking into account the fact that they are all built from the same set of variables. Then, for a target concept of size m with n variables, the bottom clause has $m \times \frac{n(n-1)}{2}$ literals. It is necessarily more specific than any hypothesis and it is covered by the target concept.

Algorithm 1 $GenerateProblemWithNearMisses$(h,m,E)

```
begin
1 $NM = \emptyset$      % the set of near-misses
2 Create the bottom clause $B$ of $\mathcal{L}_h^m$
3 For k=1 to m do      % one near-miss per concept's literal
4     Extract a negative example $e^-$ from $E$      % $E = E \setminus e^-$
5     Add to $e^-$'s a skolemised version of the bottom clause
6     While $\exists\theta$ such that $h\theta \subseteq e^-$ do
7         Remove $p_k(X_i, X_j)\theta$ from $e^-$      % the kth literal of h
8     $NM = NM \cup e^-$
9 Return $E \cup NM$
End
```

The generation procedure is given in algorithm 4.2. It takes as input a target concept h of size m, the corresponding learning problem E and outputs a new learning problem. This new learning problem has the same number of positive and negative examples as the input problem. At line 5, we see that a negative example e^- is augmented with the addition of a skolemised version of the bottom clause. Hence, it is now covered by all the hypotheses in $\mathcal{L}_h^m$ and, supposing that a solution of the learning problem exists in $\mathcal{L}_h^m$, is now turned into a positive example: for each $h \in \mathcal{L}_h^m$, we can exhibit a substitution θ, involving only the skolem constants of the added bottom clause, such that $h\theta \subseteq e^-$. It has to be modified into a negative example which is done at the lines 6 and 7. It guarantees that the new negative example is a near-miss: the longest lgg between the bottom clause and the example will differ only in the relevant literal of the target concept. When this near-miss example will be used by the TDD refinement operator to specialise the incorrect hypothesis, only the relevant literal will be produced as refinement.

The most important point of this problem generation is to note that the number of models of a hypothesis in the near-misses can only be greater than in the original negative examples: if a hypothesis covers the original negative example, it covers the corresponding near-miss and the converse is not true. This property is necessary to show that a near-miss can be added without changing the plateau profile.

At lines 6 and 7, during the computation of substitutions between the target concept and the positive example, only substitutions of the variables onto the skolem constants of the bottom clause part of the example can be found as the original example is a negative example. This property is due to the fact that a phase transition is shown in a space of hypotheses where the resulting matching problems cannot be decomposed into simpler sub-matching problems (see [4]). Therefore, the computed substitutions can only involve the skolem constants and not a mix of some constants from the original example and some skolem constants. In the next section, we show the resulting plateau profiles and discuss the behaviour of the TGT and the TDT learners on these new problems.

We ran FOIL on the same problems as before, augmented with near-misses as described above. On all of these problems, FOIL is unable to find any good approximation of the target concepts, being in the "yes", "no" or "pt" regions. As we can see in the right side of figure 3, the plateau profile does not change and a TGT learner cannot take advantage of the addition of the most informative negative examples. The problems of the "no" region exhibit a different behaviour of FOIL which cannot even output a correct theory for any of them. If we look at the rate of covered negative examples for the maximum hypothesis size, we see that the most specific hypotheses are not correct on average. These hypotheses cover the near-misses which by construction are more specific than all hypotheses but merely the ones that are on the path to the target concept. This is an illustration of the performance of FOIL as worse as random guessing on the original problems in the "no" region described in section 3.2.

The opposite behaviour is exhibited by the TDD learner PROPAL which, by construction, solves all of the problems being in the "yes", "pt" or "no" regions. The learner makes the most of the learning data by exploiting only the information provided by the near-misses to guide its search. Note that as the branching factor is reduced to one thanks to the near-misses, the target concept is exactly identified each time as opposed to evaluating the quality of the approximation on a test set as for FOIL. Interestingly, the resolution of the problems is quite fast, taking below 20 minutes on a desktop computer up to problems on the line $m = 14$ to several hours for the hardest ones.

5 Conclusion

Plateau phenomena in ILP have been studied recently in the phase transition framework and an important work has been done on identifying the criteria of success of learning algorithms [4]. The conclusion drawn from this work was that the location of the target concept with respect to the PT of the covering test was conclusive of the difficulty of the learning problems. A failure region was identified for all the tested learners, starting from the "pt" region to the beginning of the "no" region. We performed additional experiments that strengthen this result. When the top-down search is conducted in the hypothesis space that exhibits a PT in its covering test, the "yes" region acts as a plateau for the heuristic search. This is the pathological case of heuristic search, whether complete or not, as the plateau must be crossed without being able to differentiate between refinements. In such a case, the greedy TGT learner, FOIL, cannot solve any of the problems. We showed, moreover, that this criterion alone is not reliable. As a main result, we exhibited a counter-example where a TDD learning algorithm [2], supplied with near-miss examples was able to solve all problems, although the near-miss examples are still non-informative for TGT algorithms. The plateau phenomena exhibited in the PT framework is a pathological case of the TGT learners as they only rely on an evaluation function to guide their search, but it is not, *alone*, a reliable complexity measure for all learners.

References

[1] E. Alphonse. Macro-operators revisited in inductive logic programming. In *Proc. of ILP'2004*, pages 8–25, 2004.

[2] E. Alphonse and C. Rouveirol. Extension of the top-down data-driven strategy to ILP. *Proc. of ILP'2006*, 2006.

[3] H. Blockeel and L. D. Raedt. Lookahead and discretization in ILP. In *Proc. of ILP'1997*, LNAI, pages 77–84, 1997.

[4] M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning as search in a critical region. *Journal of Machine Learning Research*, 4:431–463, 2003.

[5] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In R. Myopoulos, John; Reiter, editor, *Proc. of IJCAI'1991*, pages 331–340, 1991.

[6] J. Fürnkranz and P. Flach. Roc 'n' rule learning-towards a better understanding of covering algorithms. *Machine Learning Journal*, pages 39–77, 2005.

[7] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1989.

[8] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, 1994.

[9] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[10] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[11] J. Pearl. *Heuristics*. Addison-Wesley, Reading, 1985.

[12] G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, pages 153–163. 1970.

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[14] B. D. Smith and P. S. Rosenbloom. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proc. 8th AAAI*, pages 848–853, 1990.

[15] A. Srinivasan. A learning engine for proposing hypotheses (Aleph), 1999.

[16] P. H. Winston. Learning structural descriptions form examples. In P. H. Winston, editor, *The Psychology of Computer Vision*, pages 157–209. McGraw-Hill, 1975.