A Backtracking strategy for Order-Independent Incremental Learning

Nicola Di Mauro and **Floriana Esposito** and **Stefano Ferilli** and **Teresa M. A. Basile** [1]

Abstract. Agents that exist in an environment that changes over time, and are able to take into account the temporal nature of experience, are commonly called *incremental learners*. It is widely known that incremental learning systems suffer from order effects, a phenomenon observed when differently ordered sequences of examples lead to different results. The goal of this paper is presenting INTHELEX$_{back}$, an order-independent evolution of the incremental learning system INTHELEX. A backtracking strategy is incorporated in its refinement operators, which causes a change in its refinement strategy and reflects the human behavior during the learning process. It consists in remembering the different versions of the learned theory across modifications due to new evidence. In this way the system can backtrack on a previous knowledge level when it discovers to have made a wrong choice. Experiments on an artificial dataset validate the approach in terms of computational cost and predictive accuracy.

1 INTRODUCTION

Intelligent agents should be able to work in environments that change over time. Hence, their learning component should take into account that the knowledge about the world is provided incrementally. For instance, a learning system should revise its learned knowledge as new observations are available, and it should be able to use this knowledge to carry out some task at any stage of learning. This type of learning is often named *Incremental Learning*.

The most important assumptions characterizing an incremental learning system are the following: a) the system must be able to use the learned knowledge at any step of the learning process; b) theory revision must be efficient in fitting new incoming observations; c) memory requirements must not depend on the training size. Let us recall from [5] some important definitions regarding incremental learning and order effects.

Definition 1 (Incremental Learning). A learner is *incremental* if it inputs one training experience at time, does not re-process any previous experience, and retains only one knowledge structure in memory.

This definition for incremental learning systems reflects the above three assumptions about incrementality. In particular, it is important to underline that the time taken to elaborate each observation must be almost constant in order to guarantee efficient learning and to exclude learning systems that retain competing descriptions, such as the candidate elimination algorithm [6]. It is possible to see this kind of systems as incremental hill climbing approaches to learning, affected by the training instances ordering.

[1] Dipartimento di Informatica, Università degli Studi di Bari, Italy email:{nicodimauro, esposito, ferilli, basile}@di.uniba.it

Definition 2 (Order Effect). A learner L exhibits an order effect on a training set of experiences T if there exist two or more orders of T for which L produces different knowledge structures.

The cause of this phenomenon can be discovered looking at the learning process as a search in the space of knowledge structures. In this perspective, an incremental learner chooses which path to follow from a set of possibilities (generated by new incoming instances and constrained by the previous ones) but there is no warranty that future instances will agree with this choice.

Definition 3 (Order Sensitivity). A learner L is order sensitive if there exists a training set T on which L exhibits an order effect.

Since robustness is a primary issue for any machine learning system, it is very desirable to mitigate the phenomenon of order sensitivity. This paper proposes a new general approach, based on a backtracking strategy, to decrease order sensitivity in incremental learning systems. It is important to note that what is demanded to a learner is not an exact definition of the target concept but, rather, the identification of a good approximation of the concept itself that makes the learner able to behave efficiently on future incoming instances. The goal of this new approach is to maintain the incremental nature of the learner, as stated in Definition 1, and to offer a strategy that alleviates the order sensitivity while preserving the efficiency of the learning.

As Langley pointed out in [5], there exist at least three different levels at which order effects can occur: at the level of attributes of the instances, at the level of instances, and at the level of concepts. The last two are more interesting for a deep analysis of the order effect phenomenon. In this paper we focus on the second level, where the task is to learn a concept definition from instances. In particular, we investigate the approach in the Inductive Logic Programming (ILP) framework, in which the concept definition is made up of clauses and the instances are a set of positive and negative examples describing the target concept. The problem of order effects at the level of concepts requires a further analysis and represents a future work issue.

In the following, Section 2 reports some related works about how to mitigate the order effect problem in incremental learning and Section 3 introduces the new backtracking strategy. Section 4 briefly recalls INTHELEX and describes how the backtracking strategy has been included. Finally, Section 5 reports experimental results, and Section 6 concludes with some perspective for further research.

2 RELATED WORKS

It is widely known in the Machine Learning literature that incremental learning suffers from instance order effects and that, under some orderings, extremely *poor* theories might be obtained. However, when the purpose of a machine learning system is to work in a robust manner, it is very desirable for it to be not order sensitive. Thus, a lot of approaches in order to decrease/overcome the order effect phenomenon in an incremental learner have been proposed.

The simplest way proposed to alleviate the problem is to retain all possible alternatives for a revision point, or more than one description in memory. Unfortunately, this method results very expensive from the computational complexity point of view, in both time and space. An alternative is to make strong assumptions about the nature of the target concept. The disadvantage of this method concerns the validity of such assumptions, indeed checking these representational restrictions could be computationally heavy. A more interesting approach consists in imposing constraints on the learning process by means of a background knowledge. A formal analysis of the conditions under which background knowledge reduces order effects is reported in [1], where the author identifies the causes of the order effect in the system incapacity to focus on an optimal hypothesis (when it has to choose among the current potential ones) and to keep enough information not to forget potential hypotheses. These characteristics correspond to a local preference bias that heuristically selects the most promising hypotheses. Such a bias can be viewed as a prior knowledge built into the system and can be obtained by means of additional instances provided to an order-independent system. Hence, the author reduces the problem of the right instances ordering to the problem of adding such instances (representing the prior knowledge) to an order-independent system. In this way it is proved that there are strong contingencies for an incremental learner to be order independent on some collection of instances.

Other specific strategies have been proposed to overcome the order effect problem in incremental clustering.

The NOT-YET strategy [9] exploited in COBWEB [4] tries to overcome the problem by means of a buffering strategy. The instances that cannot be added to the current cluster are stored into a buffer for future processing. The size of the buffer (i.e., the number of instances that can be "remembered") is a user-defined parameter. During the learning process, when the size of the buffer exceeds the user defined-size, the strategy elaborates the buffered instances up to this moment in order to include them in the existing clusters. Experimental results reported in [9] show that, in such an approach, there can be cases in which to induce a good cluster an high number of instances must be "bufferized" in order to be reconsidered later, and in some cases this number may amount to even 90% of the dataset.

Another approach is represented by ID5R [10], an incremental version of the ID3 batch algorithm, that builds a decision tree incrementally. The tree is revised as needed, achieving a tree equivalent to the one that the basic ID3 algorithm would construct, given the same set of training instances. In ID5R the information needed to revise the tree is maintained at each test node. This information consists of the counts of positive and negative instances for every possible value of every attribute coming from the training instances. The tree is revised changing the *position* of some (or all) test nodes according to a measure calculated on all the information (positive and negative instances) regarding such a node. Hence, the algorithm does not forget any information contained in the input data and, when it makes a choice between alternative hypotheses, it keeps enough information to be able to compose all potential competing models and, thus, to select the best one at any moment.

Unfortunately, both the above approaches do not respect at all the definition of incremental learning.

3 THE BACKTRACKING STRATEGY

The goal of a machine learning system is to simulate the human learning process, hence it is necessary to understand how to replicate, in an automatic way, some human behaviors. We think that there is a strict relation between incremental learning and the human learning process, since human learners receive information in an incremental fashion. We want to explain our approach to mitigate order effects in incremental learning by making a parallel with the identification of the right path in a maze. In a maze, arriving to a choice point, a decision about what direction to follow has to be taken. If a dead end is reached then it is necessary to go back to the last decision point and choose another direction. This process is repeated until the way out is found. In other words, the system must be provided with the ability to hypothesize the existence of another path, better than the current one, that leads to the correct solution.

From the machine learning point of view, when a learner is not able to revise the theory, due to specific *constraints* it must fulfill, it should assume that probably it had previously chosen a wrong path. In particular, an incremental learning system explores the hypotheses space with an hill-climbing approach, and this kind of myopia can be avoided providing the learner with a mechanism for backtracking over previous hypotheses. It is necessary to define some criteria, or constraints, indicating when the system must backtrack, such as *completeness*, *consistency*, and *theory minimality*.Thus, when the learner achieves a point in which it is not able to revise the current theory with respect to a new incoming instance, assuring the completeness and consistency criteria, then it could try to revise a previous one.

Furthermore, in order to backtrack on a previous theory, the system must remember at what moments it revised the theory and how it was revised. In particular, during the learning process, if at time t the system revised the theory T in T', then it should memorize that *"T' was obtained from T at time t"*. In order to perform this task, the learner memorizes a list of revisions, where each element R_t indicates that at the time t there was a revision of the theory due to incorrectness with respect to an example e_t. Furthermore, each element in this list contains the previous theory, the instance that fired the revision, and the new revised theory. Note that not the whole theories are memorized in each element, but only the specific modification made to the theory. In an ILP setting, this means that not all the clauses composing the theory are memorized but just the revised clause(s). It is important to note that this does not mean rembering/memorizing all the possible refinements for a revision point, but only that at the specific time t the theory T was revised in T' in order to fit the example e_t. This list represents a chronological trace of the knowledge changes and it is a powerful information for a theory revision system, as shown in the following.

Let us now show how the learner should reason. As already pointed out, the backtracking process is activated at time t if some constraints are violated. The learner hypothesizes that *"the current theory was obtained by choosing a wrong path"*. In order to check if this hypothesis is correct the learner restores a previous theory (say the one at time $t-k$), by using the list of revisions, and performs another revision. At this point the alternatives are: 1) re-processing all the instances arrived from time $t-k+1$ up to the time t; or 2) choosing a revision that does not violate the constraints on the instances arrived from time $t-k+1$ up to time t. Since the first alternative violates the assumption of incremental learning, the system tries to find a revision that fits all the examples seen up to the moment of the backtracking step.

At least three different situations can arise dealing with incremental concept learning in ILP. The first corresponds to the case in which the concept to be learnt is represented by a definition made up of only one clause. The second situation concerns the concepts with more than one clause in their definition. Finally, the third case corresponds to multi-conceptual learning. In this paper we give a general strategy for order effect in incremental learning, with a possible implementation dealing with the first situation above reported, and we provide an analysis of the problems that arise in the second situation.

When the system has to learn a target concept whose definition is made up of only one clause, some assumptions can be made about its behavior. Specifically, when the current theory (clause) is incomplete (resp. inconsistent) with respect to a new incoming positive (resp. negative) example, and it is not possible to revise it, then we are sure that it is always possible to backtrack and find a new theory (clause) complete and consistent with respect to all known examples.

When the target concept is defined by two or more clauses the problem of order effects becomes more difficult to handle. Suppose that the concept $\mathcal{C}$ to be learnt is made up of the clauses $C_1, C_2, \ldots, C_n$, and that we have a set of examples $\{e_{11}, \ldots, e_{1m_1}, e_{21}, \ldots, e_{2m_2}, \ldots, e_{n1}, \ldots, e_{nm_n}\}$ such that C_i explains all e_{ik}'s ($1 \leq i \leq n, 1 \leq k \leq m_i$), and C_i does not explain any e_{jh} ($j \neq i, 1 \leq h \leq m_j$). Obviously, if the learner takes as input the examples in this ordering then it should be able to learn the concept $\mathcal{C}$ expressed exactly as the disjunction of the clauses $C_1, C_2, \ldots, C_n$. Suppose that we have another random ordering of the examples, and that at time t the system learned a theory made up of only one clause C'. Furthermore, suppose that at time $t+1$ the system takes as input a new positive example e not covered by the theory and it is not able to generalize the clause C' to fit e. Hence, it is necessary to add to the theory a new clause covering the example e to make the theory complete. The problem in this situation is that the clause C' is a generalization of a set of examples that are not all, in principle, in charge of the same clause C_i. For instance, suppose that the example e belongs to the concept "black" and that the clause C' generalizes 3 examples for the concept "black" and 2 examples for the concept "white". The system is not able to generalize the clause C' with respect to the example e because C' is forced to cover also some examples for the concept "white".

A possible solution is to go back on a previous revision point by choosing another clause (revision) and to re-consider (i.e., use to revise it, if needed) all the examples acquired after this revision point. Another solution could be to define a similarity measure to extract from the set of examples covered by the clause C' a subset E of examples closer to the example e and generalize them into a new clause. In this way the clause C' is no longer forced to cover the examples of the set E, and a future generalization of C' could be possible. But all these solutions contrast with the main idea that a learning system is not required to learn exactly the correct definition. Indeed, the goal is to achieve a good approximation of the target concept evaluated, for instance, by the completeness and consistency errors on new unseen examples. Let us now analyze whether the backtracking strategy is a good solution to mitigate the order effects at least on the base cases and then extend it to more complicated problems.

4 IMPLEMENTATION IN INTHELEX

INTHELEX (INcremental THeory Learner from EXamples) is a fully incremental, multi-conceptual learning system for the induction of hierarchical theories from examples [2]. In particular, *full incrementality* avoids the need of a previous theory to be available, so that learning can start from an empty theory and from the first example; *multi-conceptual* means that it can learn simultaneosly various concepts, possibly related to each other. Moreover, it is a *closed loop system*, hence the learned theory is checked to be valid on any new example available, and in case of failure a revision process is activated upon it, in order to restore the completeness and consistency properties. INTHELEX learns theories, expressed as sets of Datalog^{OI} clauses, from positive and negative examples. Datalog^{OI} is a logic language based on the notion of *Object Identity* (*"Within a clause, terms denoted with different symbols must be distinct"*) that has the same expressive power as Datalog [8].

In a logical framework for the inductive synthesis of Datalog theories, a fundamental problem is the definition of *ideal* refinement operators. Unfortunately, when full Horn clause logic is chosen as a representation language and either θ-subsumption or implication is adopted as a generalization model, there exist no ideal refinement operators [7]. On the contrary, they do exist under the weaker, but more mechanizable and manageable ordering induced by θ_{OI}-subsumption, as proved in [3]. Note that, like θ-subsumption, θ_{OI}-subsumption induces a quasi-ordering upon the space of Datalog clauses, but this space is not a lattice when ordered by θ_{OI}-subsumption, while it is when ordered by θ-subsumption.

INTHELEX adopts a full memory storage strategy, i.e. it memorizes all the examples, thus the learned theories are guaranteed to be valid on the whole set of known examples. Besides it incorporates two inductive refinement operators, one for generalizing hypotheses that reject positive examples, and the other for specializing hypotheses that explain negative examples. When a new incoming positive example is not covered, a generalization of the theory is needed. The system chooses a clause defining the wrong concept to be generalized and tries to compute the least general generalization under object identity (lgg_{OI}) of this clause and the example. If one of the computed generalizations is consistent with all the past negative examples, then it replaces the chosen clause in the theory, or else another clause (if any) is chosen to compute the lgg_{OI}. If no clause can be generalized, the system checks if the example itself, with the constants properly turned into variables, is consistent with the past negative examples. If so, the clause is added to the theory, or else the example itself is added as an exception. When a negative example is covered, a specialization of the theory must be performed. Among the program clauses occurring in the SLD-derivation of the example, INTHELEX tries to specialize one at the lowest possible level, in order to refine the concepts that are used in the definitions of other concepts, by adding to it one or more positive literals, which can discriminate all the past positive examples from the current negative one. In case of failure, it tries to add the negation of a literal, which discriminates the negative example from all the past positive ones, to the first clause of the SLD-derivation (related to the concept of which the example is an istance). If none of the clauses obtained makes the theory consistent, then INTHELEX adds the negative example to the theory as an exception.

INTHELEX, like other incremental machine learning systems, is order-sensitive. As already pointed out, the hypothesis space ordered by the θ_{OI}-subsumption relation is not a lattice as for θ-subsumption, and hence, for any two clauses many mutually uncomparable minimal refinements might exist. When the system tries to generalize a clause, if it has more than one path to choose it cannot decide in advance which is the correct one, because it does not know future examples. As to specialization, INTHELEX uses a non-minimal operator that appends additional premises to the clause. Also in this case, if the system has many ways to specialize the clause it cannot choose the correct one.

INTHELEX$_{back}$, the improvement of INTHELEX that reduces the order effects, embeds a backtracking strategy into the two inductive refinement operators. The constraint that INTHELEX$_{back}$ imposes during learning is to have a minimal theory, and the violation of this constraint starts the backtracking revision process. In particular, when the generalization operator fails in finding a lgg_{OI} of a clause, it tries to revise a previous theory. If this second step fails, the system continues in the usual way by adding a new clause to the theory. On the other hand, the specialization operator tries to revise a previous theory only if it is not able to specialize (by adding positive or negative literals) the clause that covers the negative example.

In order to restore a previous version of the theory to be revised, INTHELEX$_{back}$ memorizes all the revisions, their type and at what time they happened. Given a set of concepts $\mathcal{C}_1, \mathcal{C}_2, \dots \mathcal{C}_n$ to be learnt, INTHELEX$_{back}$ maintains a stack of revisions for each clause of each concept $\mathcal{C}_i$. In particular, when the system performs a theory revision, this revision is "pushed" in the corresponding stack of the revised clause. In this way, the system is able to perform backtracking on previous versions of a single clause. The following is the detailed description of a generic element in the stack: $[Type, OldClause, NewClause, ExampleID]$, where $Type$ can be: 1) "adding a new clause", 2) "lgg of a clause", 3) "adding a positive exception", 4) "specialization with positive literal(s)", 5) "specialization with negative literal", or 6) "adding negative exception"; $OldClause$ is the clause revised while $NewClause$ is the new clause obtained by generalization or specialization; finally, $ExampleID$ represents the number of the example that caused the revision (i.e., the time at which the revision happened).

For a new incoming example, if the system decides to backtrack on a previous theory, then it chooses a clause C belonging to the concept to be revised and finds the corresponding revisions' stack S_C. While there are items in S_C, the system "pops" a revision $R = [T_R, OC_R, NC_R, EID_R]$ from S_C and restores the previous theory by just replacing the clause NC_R with OC_R in the theory. According to the type of the revision R the system performs one of the following tasks: 1) if the type T_R of the revision R was a lgg then it tries to find another lgg of the clause OC_R that is consistent with all negative examples older than the example EID_R, and consistent and complete with all examples newer than EID_R; 2) if the type T_R was a specialization by adding literals then it tries to find another specialization of the clause OC_R that is complete with respect to all positive examples older than the example EID_R, and consistent and complete with all examples newer than EID_R; 3) if the type T_R is the addition of an exception then the system does not perform any task and "pops" another revision; 4) when the type T_R is the addition of a new clause this means that it is not possible to revise a previous version of this clause, and hence the backtracking revision process on this clause fails. When a backtracking process fails on a clause, and there are no other clauses for that concept to be revised, then INTHELEX$_{back}$ is forced to violate the constraint of theory minimality by adding a new clause or an exception.

INTHELEX$_{back}$ fulfills Definition 1 since it does not reprocess past examples, but just stores them for ensuring correcteness. Furthermore, remembering how the theory was obtained does not mean having a set of competing hypotheses that summarize the examples, but only giving the system consciounsness of what it did.

5 EXPERIMENTAL EVALUATION

The relevance of the new proposed strategy was evaluated by comparing the learning behaviour of INTHELEX$_{back}$ with that of INTHELEX. Experiments are based on a purposely designed artificial dataset. The main reason for excluding other benchmark (or real-world) datasets is that the goal of experiments was to show the effectiveness of the proposed strategy in avoiding ordering-effect, rather than in verifying the goodness of the learning system itself. The results, in terms of accuracy of the theory, obtained by INTHELEX$_{back}$ depend on the performance of INTHELEX, and hence we may expect that the improvements obtained on artificial datasets, could be confirmed on real-world problems too. Furthermore, we needed a dataset regarding a concept whose definition was made of only one clause, since there is not still a well defined approach to solve the case of a concept expressed by many clauses. It is easy to prove (not done in this paper for lack of space) that, even on real-world datasets with the same carachteristic, when the concept to be learnt is defined by only one clause INTHELEX$_{back}$ always find it.

5.1 The problem domain

In order to investigate the ability of INTHELEX$_{back}$ in learning the correct target concept without being order sensitive, the following clause C made up of 4 variables and 5 predicate symbols was defined, representing the target concept to be learnt: $h(A) \leftarrow p_1(A,B,C), p_2(A,E,C), p_3(A,B,D), p_4(A,D,E), p_5(A,B,C)$. Note that the first variable in the literals (A) is for linkedness purposes only, since the theories learned by INTHELEX are made up of linked clauses, thus all predicates can be considered as binary.

Then 100 positive and 100 negative examples were generated at random. Specifically, each example contains 5 literals for each predicate symbol p_1, p_2, p_3, p_4, p_5, whose arguments are selected, uniformly and without replacement, from the Cartesian product of all possible pairs of 4 constants. In this way the same literal cannot occur twice. An example is considered positive if it is covered by the above clause C, and negative otherwise.

5.2 Experimental settings

INTHELEX$_{back}$ was evalueted along the same line as INTHELEX, i.e. complexity (how many clauses the theory contains) and correcteness (whether the theory contains the correct target clause) of the learned theory, and the cost (time complexity and number of theory revisions) spent to learn it.

In order to verify the order sensitivity of INTHELEX and the improvements of the new approach, independent orderings of the examples were generated from the training set. In this paper, we focused on the order sensibility of the generalization operator only. In such a way, the incremental nature of the system is preserved and only one type of search is analyzed. Hence, the training set was built by putting all the 100 negative examples at the beginning, followed by all the positive ones. In this way, starting from an empty theory, the system will learn and revise the theory starting from the first positive example, using the negative ones as a further *search bias* (i.e., a constraint on the way a system explores the clauses in the search space).

5.3 Experimental results

In the first experiment INTHELEX was run on 34 different orderings of the training set. Table 1 reports the results. The first row reports all cases, the second row indicates the cases in which INTHELEX learnt a theory containing the target clause, while the third row represents the cases in which it was not able to learn a correct theory.

The fact that the learned theory contains the correct clause but also other clauses means that the correct clause was learnt at the end of learning process. In this case it is possible to use a cleaning process to eliminate all the clauses subsumed by the correct one. The first column indicates the runtime expressed in milliseconds, the second column the number of clauses composing the theory, and the third column the number of lgg_{OI} performed, all averaged on the number of orderings (reported in the last column). INTHELEX is able to find the correct solution in 21 cases out of 34. The high values in the third row indicate the difficulty of the system to learn with a bad ordering of examples.

Table 1. INTHELEX performances

	Time	Clauses	Lgg	Run
All	230895	13.94	34	34
Correct	89970	8.71	21.14	21
Wrong	461733.8	22.38	55.77	13

Table 2 reports the improvement of INTHELEX_{back} on the same set of orderings. Results reveal that it always learns the correct clause in 41 seconds on average. The fourth column indicates that during the learning process the system made 7.65 mistakes (corresponding to the number of needed backtrackings). The number of "pops" for each backtracking requested by the system is $1.22 (= 9.34/7.65)$ and the number of revisions (i.e., number of "push") is 12.46.

Table 2. INTHELEX_{back} performances

Time	Clauses	Lgg	Backtracking	Push	Pop
41569.2	1	11.46	7.65	12.46	9.34

When run on 100 orderings rather than on 34, INTHELEX_{back} learned always the correct target concept after 13.85 positive examples on average (with minimum of 6 and a maximum of 82).

To have an idea of the behaviour of INTHELEX_{back} in using both generalization and specialization operators, the same training set was used and 34 different orderings were built by mixing positive and negative examples. In this case, the time complexity of INTHELEX was better (211913.5) and the number of clauses composing the theory was almost the same (13.68); the theory was obtained by means of 37.53 generalizations, 21.2 specializations and by adding 12.5 negative exceptions. However, in this case INTHELEX found the correct solution in only 14 cases out of 34; this indicates that putting all the negative examples at the beginning is a profitable bias. The behaviour of INTHELEX_{back} is interesting. It always finds the correct solution, spending almost the same time (42586.5), by means of fewer generalizations (6.8), backtracking (only 3.47), push (8.18) and pop (6), and by using 0.38 positive specializations. Furthermore executing INTHELEX_{back} on 100 orderings rather than on 34 we found that it converges rapidly to the correct solution: after 17.1 examples on average (with minimum of 12 and a maximum of 26).

It could be interesting to analyze the minumum amount of examples required by the system to correctly learn the theory. Hence, the above training sets were corrupted by dropping a progressively higher percentage of negative and positive examples, starting from 10% up to 95%. Table 3 reports the results, showing that the time complexity decreases according to the number of examples while the number of the other operations requested by the system are the same until dropping from the dataset 80% of examples. Only above that percentage the figures neatly changed, i.e. when the training set was reduced to only 20 (10 positive and 10 negative) and 10 (5 positive and 5 negative) examples. Note that the system was able to grasp the correct concept until more than half of the examples were removed from the training set.

Table 3. INTHELEX_{back} results

Problem	Time	Lgg	Mist.	Push	Pop	Correct
90	44940	11.19	7.63	12.19	9.38	100%
80	42474	10.82	7.37	11.82	8.94	100%
70	44411	10.85	7.27	11.85	9.12	100%
60	32039	11.24	7.59	12.24	9.30	100%
50	29967	11.69	8.00	12.69	9.99	100%
40	36334	13.90	9.94	14.90	12.45	98%
30	26575	11.55	7.56	12.55	9.83	98%
20	22436	10.95	6.85	11.95	9.13	78%
10	8305	7.69	3.93	8.69	5.25	25%
5	470	3.74	0.80	4.74	0.89	0%

6 CONCLUSION AND FUTURE WORKS

This paper[2] presented a backtracking strategy for mitigating order effects in incremental learning, that was implemented in INTHELEX_{back}, a modification of INTHELEX. Preliminary results show that the strategy reaches good performance when compared to an order-sensitive learner. Future work will concern a more accurate evaluation of the strategy using also the specialization operator of INTHELEX, only mentioned in this paper. Furthermore, an investigation is ongoing on how to manage the case of learning a concept whose definition is made up of more than one clause, and the most complicated case of multiple concept learning task.

REFERENCES

[1] A. Cornuéjols, 'Getting order independence in incremental learning', in *ECML93*, ed., P. Brazdil, volume 667 of *Lecture Notes in Artificial Intelligence*, pp. 196–212. Springer Verlag, (1993).

[2] F. Esposito, S. Ferilli, N. Fanizzi, T.M.A. Basile, and N. Di Mauro, 'Incremental multistrategy learning for document processing', *Applied Artificial Intelligence Journal*, **17**(8/9), 859–883, (2003).

[3] F. Esposito, A. Laterza, D. Malerba, and G. Semeraro, 'Locally finite, proper and complete operators for refining datalog programs', in *ISMIS96*, ed., Z.W. Rás M. Michalewicz, volume 1079 of *Lecture Notes in Artificial Intelligence*, pp. 468–478. Springer Verlag, (1996).

[4] D. H. Fisher, 'Knowledge acquisition via incremental conceptual clustering', *Machine Learning*, **2**, 139–172, (1987).

[5] P. Langley, 'Order effects in incremental learning', in *Learning in humans and machines: Towards an Interdisciplinary Learning Science*, eds., P. Reimann and H. Spada, Oxford: Elsevier, (1995).

[6] T. Mitchell, 'Generalization as search', in *Artificial Intelligence*, volume 18, 203–226, (1982).

[7] S.-H. Nienhuys-Cheng and R. de Wolf, *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*, Springer Verlag, February 1997.

[8] G. Semeraro, F. Esposito, D. Malerba, N. Fanizzi, and S.Ferilli, 'A logic framework for the incremental inductive syntesis of datalog theories', in *LOPSTR98*, ed., N.E. Fuchs, volume 1463, pp. 300–321. Springer Verlag, (1998).

[9] Luis Talavera and Josep Roure, 'A buffering strategy to avoid ordering effects in clustering', in *ECML98*, pp. 316–321, (1998).

[10] P. E. Utgoff, 'Incremental induction of decision trees', *Machine Learning*, **4**, 161–186, (1989).

[2] This work was partially funded by the EU sixth framework programme VIKEF (Virtual Information and Knowledge Environment Framework).