Avoiding Order Effects in Incremental Learning

Nicola Di Mauro, Floriana Esposito, Stefano Ferilli, and Teresa M.A. Basile

Department of Computer Science, University of Bari, Italy
{ndm, esposito, ferilli, basile}@di.uniba.it

Abstract. This paper addresses the problem of mitigating the order effects in incremental learning, a phenomenon observed when different ordered sequences of observations lead to different results. A modification of an ILP incremental learning system, with the aim of making it order-independent, is presented. A backtracking strategy on theories is incorporated in its refinement operators, which causes a change of its refinement strategy and reflects the human behavior during the learning process. A modality to restore a previous theory, in order to backtrack on a previous knowledge level, is presented. Experiments validate the approach in terms of computational cost and predictive accuracy.

1 Introduction

In many situations, intelligent systems are bound to work in environments that change over time. In these cases, their learning component, if any, must take into account that the available knowledge about the world is provided over time. For instance, a learning system should revise its learned knowledge when new observations are available, while still being able to provide at any stage of the learning process such a knowledge in order to carry out some task. This kind of learning is often named *Incremental Learning*.

As pointed out in [1], the three most important assumptions characterizing an incremental learning system are: a) "it must be able to use the learned knowledge at any step of the learning"; b) "the incorporation of experience into memory during learning should be computationally efficient" (theory revision must be efficient in fitting new incoming observations); and, c) "the learning process should not make unreasonable space demands, so that memory requirements increase as a tractable function of experience" (memory requirements must not depend on the training size).

It is clear that, in the incremental learning setting, the shape of the learned theories can be strongly influenced by the order in which the examples are provided to the system and taken into account by it. This paper carries on the work reported in [2], facing the problem of making a learning system insensitive to examples ordering. While the aim of that work was a preliminary check of the performance of the proposed technique (and hence it focused on a specific feature of the system, namely the generalization operator), here the research is extended to take into account the general case, in which positive and negative examples, and consequently generalization and specialization operations, can be interleaved. As reported in [1], we can define incremental learning as follows:

S. Bandini and S. Manzoni (Eds.): AI*IA 2005, LNAI 3673, pp. 110–121, 2005.

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

The first condition, *"L inputs one training experience at time"*, avoids to consider as incremental the learning algorithms exploited by *batch* learning systems, that process many instances at at time, by simply storing the instances observed so far and running the method on all of them. The second condition, *"does not re-process any previous experience"*, places a constraint on the learning mechanism itself, by ruling out those systems that process new data together with old data in order to come up with a new model. in that it can process each experience only once. The important idea is that *the time taken to process each experience must remain constant* or nearly so with increasing numbers, in order to guarantee efficient learning of the sort seen in humans. Finally, the third condition, *"retains only one knowledge structure in memory"*, leaves out algorithms like CE [3], since it requires to memorize exactly one definition for each concept. CE processes instances one at a time and does not need to reprocess them. However, it retains in memory a set of competing hypotheses summarizing the data, that can grow exponentially with the number of training items, and it reprocesses these hypotheses upon incorporating each training case.

Systems whose behaviour falls in Definition 1 can be seen as incremental hill climbing approaches to learning, affected by the training instances ordering. The cause of such a phenomenon can be discovered by 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 2 (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, mitigating the phenomenon of order sensitivity is very desirable. The approach to decreasing order sensitivity in incremental learning systems, proposed in this paper, is based on a backtracking strategy. It aims at being able to preserve the incremental nature of the learner, as stated in Definition 1, and to offer a strategy that alleviates the order sensitivity while preserving efficiency of the learning process. Indeed, a generic learner is not required to provide an exact definition of the target concept but, rather, to identify a good approximation of the concept itself that makes it able to behave efficiently on future incoming instances.

As pointed out in [1], 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. This paper focuses 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 further analysis and represents a future work issue.

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, since 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 [4]. The Author identifies the causes of the order effect in the system's 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 correct 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 [5] exploited in COBWEB [6] 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 elaborations. 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 [5] 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 the ID5R algorithm [7], 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 in order to compose all potential competing models and, thus, to select the best one at any moment.

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, as confirmed in learning naive physics experiments [8]. Our approach to mitigate order effects in incremental learning can be explained by making a parallel with the identification of the exit path in a maze. In a maze, arriving to a choice point, a decision about which 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 to fit new incoming observations, 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 could be avoided providing the learner with a mechanism for backtracking over previous hypotheses. In order to do this, 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, fulfilling 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 it is not necessary to memorize the whole theories 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.

After having introduced the main idea of the backtracking approach to avoid order effects, and how to make it effective by memorizing a chronology of the knowledge revisions, let us now show how the learner should reason. As already pointed out, the backtracking process is activated at time t if any constraint is 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 there are two alternatives: 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, and a possible implementation dealing with the first situation above reported. Furthermore, we provide an analysis of the problems that arise when we consider the order effect in the second situation.

3.1 One Target Concept Made Up of a Single Clause

When the system has to learn a target concept $\mathcal{C}$ whose definition is made up of only one clause, we can make some assumptions 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.

3.2 One Target Concept Made Up of More Than One Clause

When the target concept is defined by two or more clauses the problem of order effects becomes more difficult to handle. Given the concept $\mathcal{C}$ to be learnt made up of the clauses $C_1, C_2, \ldots, C_n$, and 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 $(1 \leq i \leq n \wedge 1 \leq k \leq m_i)$, and C_i does not explain any e_{jh} $(j \neq i \wedge 1 \leq h \leq m_j)$. 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. This second solution can be viewed as an *incremental clustering in first-order logic*. 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 to 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 [9]. 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$^{\mathrm{OI}}$ clauses, from positive and negative examples. Datalog$^{\mathrm{OI}}$ 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 [10].

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 representation language and either θ-subsumption or implication is adopted as generalization model, there exist no ideal refinement operators [11]. On the contrary, they exist under the weaker, but more mechanizable and manageable ordering induced by θ_{OI}-subsumption, as proved in [12]. Note that, like θ_{OI}-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. When our knowledge is inconsistent with new observations, we may ignore the inconsistency, hoping that it is insignificant or accidental, and retain our knowledge unaltered, or we may use the *evolutionary* approach[1], defined in [13], by making incremental modifications to the appropriate part of our knowledge. The strength of an incremental *full memory* method lies in its ability to use all the original facts for guiding the process of modifying and generalizing knowledge structures, and selecting alternative solutions, and to guarantee the *completeness* and *consistency* of the modified knowledge with all the examples. This is not in contrast with Definition 1, since original examples are only used to guarantee the completeness and consistency requirements of ILP systems.

INTHELEX 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 ($\mathrm{lgg}_{\mathrm{OI}}$) 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 a new clause (if it exists) is chosen to compute the $\mathrm{lgg}_{\mathrm{OI}}$. 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. Such a clause is added to the theory, or else the example itself is added as an exception to the theory. 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

[1] That is opposite opposite to the *revolutionary* approach to throw away this piece of knowledge altogether and develop another one from scratch.

makes the theory consistent, then INTHELEX adds the negative example to the theory as an exception.

Naturally, like other incremental machine learning systems, INTHELEX 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. For the specialization INTHELEX uses a non-minimal specialization operator by appending 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$_{\mathrm{back}}$, the improvement of INTHELEX that reduces the order effects, embeds a backtracking strategy into the two inductive refinement operators. The constraint imposed by INTHELEX$_{\mathrm{back}}$ during learning is to have a minimal theory, and the violation of this constraint starts the backtracking revision process. In particular, as reported in Algorithm 1, when the generalization operator fails in finding a lgg$_{\mathrm{OI}}$ 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 literals, the clause that covers the negative example. The choice to revise a previous theory after step 'a1' is justified by the theory minimality constraints; conversely, in the other case, after step 'a2', we preferred to have a theory obtained without negative literals since once added they cannot be ever removed.

Algorithm 1 INTHELEX$_{\mathrm{back}}$ algorithm

Given: a theory T and a source of examples
acquire the next example e
if e is positive and not covered by T **then**
a1) try to find the lgg$_{\mathrm{OI}}$ of a clause in T; otherwise, b1) *try to revise a previous theory*; otherwise, c1) try to add a new clause to T; otherwise, d1) add a positive exception to T
if e is negative and covered by T **then**
a2) try to specialize a clause of T by adding positive literal(s); otherwise, b2) *try to revise a previous version of* T; otherwise, c2) try to specialize a clause of T by adding negative literal(s); otherwise, d2) add a negative exception to T

In order to be able to restore a previous version of the theory to be revised, INTHELEX$_{\mathrm{back}}$ memorizes all the theory revisions. Futhermore, it memorizes also the type of the revision and at what time it happened. Given a set of concepts $C_1, C_2, \ldots C_n$ to be learnt, INTHELEX$_{\mathrm{back}}$ maintains a stack of revisions for each clause of each concept C_i. In particular, when the system performs a theory revision, this revision is "pushed" in the correspondig 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. While there are items in the revisions' stack S_C of 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 T_R of the revision R, the system performs one of the following tasks:

a) if T_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;
b) if 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;
c) if T_R was addition of an exception then the system does not perform any task and "pops" another revision;
d) when T_R was 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 $\text{INTHELEX}_{\text{back}}$ is forced to violate the constraint of theory minimality by adding a new clause or an exception.

5 Experiments

The relevance of the new proposed strategy was evaluated by comparing the learning behaviour of $\text{INTHELEX}_{\text{back}}$ to that of INTHELEX, in which it was embedded. In the particular context of this work, experiments are based on a purposely designed artificial dataset. The main reason for excluding real-world datasets is that we want to evaluate the effectiveness of the proposed strategy in avoiding ordering-effect, rather than the goodness of the learning system itself. Furthermore, the results obtained by $\text{INTHELEX}_{\text{back}}$ depend on the performance of INTHELEX and hence we may expect that possible improvements obtained on artificial datasets, could be confirmed on real-world problems too. Last but not least, we needed a dataset for a concept whose definiton 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.

5.1 The Problem Domain

In order to investigate the ability of INTHELEX$_{\text{back}}$ in learning the correct target concept without being order sensitive, the following learning problems P_1 and P_2 were defined. Both regard a target concept represented by a clause made up of 3 variables and 4 predicate symbols:

$C : h(A) \leftarrow p_1(A, D, C), p_2(A, B, C), p_3(A, D, B), p_4(A, B, C).$

Note that variable A is used just for linkedness purposes, since the theories learned by INTHELEX are made up of linked clauses only, thus all predicates can be considered as binary. Then, for each problem, 100 positive and 100 negative examples were generated at random. Specifically, each example contains 4 literals for each predicate symbol p_1, p_2, p_3, p_4, whose arguments are selected, uniformly and without replacement (so that the same literal cannot occur twice), from the set of all possible pairs of 4 constants for problem P_1, and 6 constants for problem P_2. Each example was considered as positive if it is covered by clause C, negative otherwise. Specifically, the probability that a generated example is positive is equal to $497/597 = 0.83$ for problem P_1, while it is equal to $100/1070 = 0.09$ for problem P_2. Thus, for problem P_1 it was necessary to generate 497 positive examples in order to generate 100 negative ones, while for problem P_2 970 negative examples had to be generated to reach 100 positive examples. Hence, in P_2 we expect a number of specializations higher than in P_1.

INTHELEX$_{\text{back}}$ has been evaluated along the same parameters 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 cost (time complexity and number of theory revisions) spent to learn it. To highlight the order sensitivity of INTHELEX and the improvements of the new approach, independent orderings of the examples were generated from the training set.

5.2 Experimental Results

In the first experiment INTHELEX was run on 100 different orderings of the training set. Table 1 reports the averaged results. For each problem, P_1 and P_2, the row labeled *Correct* refers to the cases in which INTHELEX learnt a theory containing the target clause, the row labeled *Wrong* reports the cases in which it was not able to learn a correct theory, while the row labeled *All* represents all cases. For problem P_2 only "All" is reported, since in only one case INTHELEX was unable to learn a correct theory. The columns indicate, respectively: the runtime (expressed in milliseconds), the number of clauses composing the theory, the number of lgg$_{\text{OI}}$ performed, the number of specializations (positive + negative), and the number of exceptions (positive + negative), all averaged on the number of orderings, reported in the last column. Whenever the learned theory contains other clauses in addition to the correct one, this means that the correct clause was learnt at the end of learning process. In this case, a 'cleaning' process that eliminates all the clauses subsumed by the correct one can be used.

For problem P_1, INTHELEX is able to find the correct solution in 55 cases out of 100. The high values in the row "Wrong" indicate the difficulty of the

Table 1. INTHELEX performance

Problem		Time	Clauses	Lgg	Specializat.	Excepti.	Run
	(All)	44160.5	4.93	16.95	4.39 + 1.54	0 + 19.01	100
P_1	(Correct)	23961.64	4.07	14.78	4.4 + 1.33	0 + 9.47	55
	(Wrong)	68848	5.98	19.6	4.38 + 1.8	0 + 30.67	45
P_2	(All)	19128.9	5.98	11.54	5.37 + 6.14	0 + 2.39	100

Table 2. $\text{INTHELEX}_{\text{back}}$ performance

	Time	Clauses	Lgg	Pos.Spec.	Backtracking	Push	Pop
P1	13376.3	1	6.47	0.29	2.29	7.76	4.04
P2	2718.8	1	4.25	0.44	1.87	5.69	2.95

system to learn when the ordering of examples is bad. Table 2 witnesses the performance improvement obtained by using the modified system $\text{INTHELEX}_{\text{back}}$ on the same set of orderings. Results reveal that it always learns the correct clause spending on average less time than INTHELEX. The fourth column indicates the number of backtrackings needed by the system during the learning process, that corresponds to the number of mistakes it made. Specifically, the number of revisions corresponds to the number of "push" operations. Conversely, the number of "pop" operations corresponds to the wrong refinements that were withdrawn by the system for each backtracking. It is interesting to note that the number of withdrawn refinements is on average 1.76 (= 4.04/2.29) on a total of about 7 (6.47 + 0.29) refinements performed in problem P_1 (1.58 (= 2.95/1.87) on about 5 (4.25 + 0.44) refinements for problem P_2, respectively), indicating that backtracking does not retract too many choices, which is important since going too deep into the refinements stack could cause an increase in computational time.

Running $\text{INTHELEX}_{\text{back}}$ on the 100 orderings we found that it always learns the correct target concept for problem P_1 (respectively P_2) after 20.11 (respectively 13.61) examples on average, with minimum of 8 (respectively 6) and a maximum of 37 (respectively 37). These results show the capability of the system to converge rapidly to the target concept. Note that using higher values of the parameters (number of constants, variables, predicates and literals) to generate the examples, than in this experimental setting, has not proved to affect effectiveness of the system, but only its efficiency (more time, revisions and backtraking). For instance, increasing the number of constants and/or literals changes the probability that the clause C covers the examples, while increasing the number of variables and/or predicates causes a growing of the search space. What one expects is a greater effort in finding the correct concept, and not that in these conditions the system does not find it.

6 Conclusions and Future Works

This paper presented a backtracking strategy for mitigating order effects in incremental learning, that was implemented in $\text{INTHELEX}_{\text{back}}$, a modification of

the first-order learning system INTHELEX.Experimental results on a purposely designed dataset show that the system, modified with the new approach, achieves better performance with respect to the basic version in all metris. Future work will concern an investigation on how to manage the case of learning a concept whose definition is made up of more than one clause, and the more difficult case of a multiple concept learning task.

References

1. Langley, P.: Order effects in incremental learning. In Reimann, P., Spada, H., eds.: Learning in humans and machines: Towards an Interdisciplinary Learning Science. Elsevier (1995)
2. Di Mauro, N., Esposito, F., Ferilli, S., Basile, T.A.: A backtracking strategy for order-independent incremental learning. In de Mantaras, R.L., ed.: Proceedings of ECAI04, IOS Press (2004)
3. Mitchell, T.: Generalization as search. Artificial Intelligence **18** (1982) 203–226
4. Cornuéjols, A.: Getting order independence in incremental learning. In Brazdil, P., ed.: Proceedings of ECML93. Volume 667 of LNAI., Springer (1993) 196–212
5. Talavera, L., Roure, J.: A buffering strategy to avoid ordering effects in clustering. In: Proceedings of ECML98. (1998) 316–321
6. Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning **2** (1987) 139–172
7. Utgoff, P.E.: Incremental induction of decision trees. Machine Learning **4** (1989) 161–186
8. Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Conceptual change in learning naive physics: The computational model as a theory revision process. In Lamma, E., Mello, P., eds.: Advances in Artificial Intelligence (AI*IA99). LNAI, Springer (1999) 214–225
9. Esposito, F., Ferilli, S., Fanizzi, N., Basile, T., Di Mauro, N.: Incremental multistrategy learning for document processing. Applied Artificial Intelligence Journal **17** (2003) 859–883
10. Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., S.Ferilli: A logic framework for the incremental inductive synthesis of datalog theories. In Fuchs, N., ed.: Proceedings of LOPSRT97. Volume 1463 of LNCS., Springer (1998) 300–321
11. Nienhuys-Cheng, S.H., de Wolf, R.: Foundations of Inductive Logic Programming. Volume 1228 of LNAI. Springer (1997)
12. Esposito, F., Laterza, A., Malerba, D., Semeraro, G.: Locally finite, proper and complete operators for refining datalog programs. In Rás, Z., Michalewicz, M., eds.: Proceedings of ISMIS96. Volume 1079 of LNAI., Springer (1996) 468–478
13. Michalski, R.S.: Knowledge repair mechanisms: Evolution vs. revolution. In: Proceedings of ICML85, Skytop, PA (1985) 116–119