Maintenance of Discovered Knowledge

Michal Pěchouček, Olga Štěpánková, and Petr Mikšovský

{pechouc,step,miksovsp}@labe.felk.cvut.cz

The Gerstner Laboratory for Decision Making and Control
Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University
Technická 2, CZ 166 27, Prague 6

Abstract. The paper addresses the well-known bottleneck of knowledge based system design and implementation – the issue of knowledge maintenance and knowledge evolution throughout lifecycle of the system. Different machine learning methodologies can support necessary knowledge-base revision. This process has to be studied along two independent dimensions. The first one is concerned with complexity of the revision process itself, while the second one evaluates the quality of decision-making corresponding to the revised knowledge base. The presented case study is an attempt to analyse the relevant questions for a specific problem of industrial configuration of TV transmitters. *Inductive Logic Programming* (ILP) and *Explanation Based Generalisation* (EBG) within the *Decision Planning* (DP) knowledge representation methodology, have been studied, compared, and tested on this example.

1 Introduction

Configuration, as a specific kind of decision making, is defined [9] as a problem of assembling basic elements of the considered system in such a way that internal logic constraints are not violated. We address the issue of gradual evolution of configuration meta-knowledge that forms a knowledge base of a knowledge-based system developed for a Czech TV producer. The considered production undergoes some changes during its life cycle, e.g. due to modification of the product range. This transition has to be reflected in the knowledge base of the system, of course. Some machine learning techniques can support corresponding evolution/revision of the knowledge base. We have applied *Inductive Logic Programming* (ILP) and *Explanation Based Generalisation* (EBG) complemented by *Decision Planning* (DP) knowledge representation methodology. Our experiments and analysis show that both methodologies exhibit different behaviour with respect to two angles introduced for characterisation of the revision processes. This behaviour should be taken into account when choosing the proper tool for support of knowledge-base evolution.

Inductive logic programming (ILP) enriched the repository of available machine learning methods in the 90ties. Its goal is to induce knowledge in the form of a general

J.M. Zytkow and J. Rauch (Eds.): PKDD'99, LNAI 1704, pp. 476-483, 1999.

logic program using in its body predicates from the background knowledge defined in advance. This is an obvious difference when compared to knowledge represented by a decision tree or a decision list, for example. In ILP, the language of the first order logic is used (1) to describe the training examples, (2) to express the induced knowledge, and (3) to formulate background knowledge, which is a natural part of the problem definition. Though ILP methods work with a significantly richer language than the attribute valued propositional language of the classical ML methods, both approaches share intuition and some basic paradigms. Number of different ILP systems have been developed and implemented recently [2]. Our experiments have been conducted in the systems FOIL [7] and KEPLER [10].

Decision Planning is a declarative knowledge representation methodology based on proof planning, a well-known theorem proving technique. This technique has been successfully used for formalisation of the process of industrial configuration (see [5, 6]). The meta-level inference knowledge is captured here by means of an oriented graph called *decision graph*. The decision graph describes in meta-terms a *decision space*, a space of all legal configurations. We distinguish between the *abstract level* of the decision graph, where only the abstract inference knowledge (strategic knowledge) is specified and the *specific level* with all the context-related pieces of knowledge incorporated. A knowledge engineer specifies the strategic knowledge within the abstract level of the decision plan and an EBG (Explanation Based Generalisation) [3, 4] machine learning algorithm deduces the specific level of the decision plan from the object-level knowledge. The basic EBG procedure starts from an attempt to prove that a concept example is positive within a complete theory, the system is aware of. The constructed proof (explanation) is then generalised and appended to the theory so that a new, better definition of the concept is learned.

We have thoroughly tested both methodologies for creating and maintaining the inference meta-knowledge to be used in the process of configuration [8]. Meta-knowledge is supposed to embody original domain knowledge from the field. To generate this meta-knowledge, we have analysed a set of all positive examples of legal variations with the aim to find knowledge that will cover exactly the given set of legal variations. This set remains fixed for a certain period. Machine learning techniques are used to elicit the relevant knowledge. The learned knowledge has to be sound and complete with respect to the present set of all legal variants representing a noise-free training set. From time to time, the set of all legal variants is extended, while the task domain remains fixed. This is a signal for the knowledge revision/update - a central point of this case study.

2 Meta-knowledge Lifecycle

Suppose that the original theory **T** covers a set of all positive examples **E**. Let the set of positive examples be extended by a new set **S** such that $\mathbf{E} \cap \mathbf{S}=\varnothing$. The theory **T** has to be updated/revised in such a way that the resulting theory **T1** covers $\mathbf{E} \cup \mathbf{S}$. To do so two alternative approaches can be taken:

- STRONG UPDATE re-computes the entire inference knowledge base considering a new updated field theory (i.e. the current field theory enriched by the recent change) as a knowledge induction parameter. Strong update is the result of application of the knowledge extraction process on the full set $\mathbf{E} \cup \mathbf{S}$.
- WEAK UPDATE re-computes just the relevant parts of the inference knowledge base in order to ensure that the result stands for a sound and complete representation for the updated field theory.

Most often it is much simpler to generate a weak update then the strong one. On the other hand, it is not clear what efficiency will exhibit the system using the changed (weakly updated) knowledge base. Weak update patches the local requirements of the decision space but it does not consider its overall shape. That is why weak update is computationally less expensive, but the resulting space of configuration-like meta-level knowledge can be expected to be quite messy. Consequently, there can appear inefficient consultations in such a space often.

When deciding between strong and weak update the efficiency of the resulting system has to be considered simultaneously with requirements for creation of the updated system. This last criterion should meet the demands of the dynamics of the application domain; i.e. estimated frequency of updating and its effect on the quality of the decision space described by inference knowledge. When the revision is not required very often and strong update is not extremely time consuming then the strong update is generally recommended. When the field theory changes instantly one has to bear in mind that we might be lacking resources or time for the strong update.

In order to illustrate this we may view the problem from purely AI point of view. Let us consider a simplified meta-knowledge decision space in a form of a decision tree, where each product configuration is a branch from a root of the tree to a certain leaf node (goal). Here, a weak update means appending a simple branch to the root of the tree. By doing so the numbers of nodes considered as well as the branching factor of the new state space increases. In this way the knowledge hidden within the original decision tree loses its significance. Using a tree with an increased branching factor or with more decision nodes results in higher memory requirements, moreover it makes the process of state space search slower and more difficult. That is why we try to update the decision space in such a way that (1) needed time and (2) the increase of the state space size and of its branching factor are as little as possible.

Neither pure weak update, nor pure strong update guarantees optimal knowledge base maintenance. Rather than implementing a *'middling'* update we seek for a compromise by playing around with frequencies of both weak and strong updates. If the weak update does not result in significant decrease of efficiency, the strong update can be done only occasionally (after several steps of weak updates). Need for strong updates and their frequency has to be determined with respect to the conditions of the application domain. We will compare experimental results concerning revision processes in DP environment with the theoretical estimates computed for Prolog programs resulting from the ILP analysis. These experiments try to answer the following questions: "Which knowledge is easier to maintain? Is it knowledge, the system formulated automatically (this is what ILP does) or knowledge elicited and expressed by a knowledge expert (the case of DP)? What is the efficiency of the system after a sequence of weak updates? "

2.1 Inductive Logic Programming Approach

ILP is applied in this context to induce inference meta-knowledge from object-level data. Our ILP analysis starts from pure data without any expert inference knowledge. The set of positive examples represents all legal variants (valid combinations) of input transmitter parameters to be covered by induced rules representing the meta-knowledge. The negative examples are created using closed world assumption. The induced set of rules should provide significant compression of positive examples; i.e. it should be as small as possible. The rules represent the inference knowledge needed during the transmitter configuration consultation. Their utilisation is straightforward – any question concerning legality of a certain variant of a product is answered using a standard Prolog query. Moreover, these rules are further analysed to optimise number of questions that have to be asked during the process of configuration.

The induced Prolog program is used whenever the question about legality of a specific variant arises. We know the exact number of different relevant queries, which can appear. Let us denote this number N – it is the size of the considered task domain. This number is constant until a new possible value for one of the attributes is added (its domain is extended). Let us concentrate on the case when all attributes have fixed domains. Then N is fixed and any member of the considered space can become an issue. That is why we are interested in average case complexity of the program measured in terms of the average number of attempted unifications.

Let us try to predict the change of this complexity during the process of gradual weak updates when each new positive example is added as an exception in front of the actual program. Let us start with the program P induced from positive examples E^+ and negative examples E^- (both sets E^+ and E^- together have just N elements). The program P works with average complexity p, it has d rules and its structure is not recursive. At the first step, one single example f is shifted from the set E^- into the set E^+. What will be the complexity of the program $P(1)$ that results from the weak update of P after this single step? Let us denote this complexity by $C(p,1)$.

The program $P(1)$ will generate the answer for most members of the task domain attempting just one more unification then the original program P. The only exception will be the single example f - this will be answered in a single unification. Thus we can estimate the upper limit for the considered update as follows

$$C(p,1) \le \frac{(Np - d + 1 + N - 1)}{N} = p + 1 - d/N \tag{1}$$

By induction we can prove that after a sequence of k gradual weak updates (each removing a single example from E^-) the complexity of the resulting program $P(k)$ is

$$C(p,k) \le \frac{(1 + 2 + \ldots + k + Np - kd + (N - k)k)}{N} = p + k - (k^2 - k + 2kd)/2N \tag{2}$$

This bound is strongly influenced by the compression coefficient d/N, which is in our case much smaller than 1/5. For any $k < N/2$ in this range holds, that the degradation of performance of $P(k)$ is within the interval $< k/2, k >$ and the increase of average case complexity after a sequence of k weak revisions is linear in k.

Assertion 1*:** Degradation of performance after ***k weak updates is acceptable provided ***k*** is a fraction of average complexity ***p*** of the original program.

The induced rules are not used to compress the set of valid variants of transmitters, only. A specific algorithm was implemented to present valid process of configuration consultation using the induced rules. For implementation reasons the task has been "inverted" and the positive and negative examples were exchanged so that no restrictions are placed on the configuration parameters at first. During consultation the state space of possible variants is cut wrt disabled combinations of parameters and engaged values. The length of configuration depends on the process of consultation as the algorithm skips all decision nodes offering a single option, only. Experiments verified that time complexity of the algorithm is linear wrt the number of rules used within the consultation. This is why the minimal number of rules is desired.

Applying strong or weak update strategies carries out the maintenance of inference meta-knowledge. Whereas, the weak update strategy can be applied as a fast strategy the time complexity of the corresponding configuration consultation algorithm grows inevitably. Though this degradation can be characterised as graceful, see equations (1) and (2), it can eventually exceed acceptable limits. Moreover, the weak update strategy is applicable only if we need to add a new legal variant within the fixed task domain. In the opposite case the strong update is inevitable.

The strong update is performed by the ILP analysis, which induces new rules from the entire data set. This strategy minimises the number of rules to be used within consultation but it is more time consuming. The strong update can significantly improve efficiency of the considered system. This is best visible on an extreme case of strong update applied to 2658 examples and resulting in 42 rules [8]. Since the time complexity is proportional to the size of the program we can claim that complexity has been decreased and efficiency improved about 63 times due to strong update.

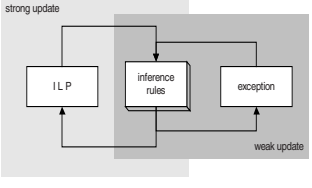

Figure 1 – ILP knowledge evolution lifecycle

We have failed to produce a rigorous time requirement analysis for the strong update (it took about 2 minutes). According to our observations, time necessary for strong update depends on the internal similarity of examples. It took considerably more time to analyse a small set of utterly unrelated examples than a bigger set of similar clusters of examples. The point we are trying to make here is that not only the frequency of strong versus weak updates is what matters. It is highly recommended to structure the incoming examples in clusters with consideration of some internal logic. This is the case of real data as the requirement for strong update arises usually when a new product with a number of variations is introduced.

2.2 Decision Planning Solution

Decision planning takes constant number of steps for any consultation within the fixed task domain. This is due to the topology of the decision plan (the abstract decision

plan) which remains fixed in such a case. No matter how many examples are covered, the user has to be taken through an identical decision path, i.e. through the same number of decision nodes. Requested time does not in practice depend on the cardinality of the set of field theory examples, which represent the set of all positive examples. Checking preconditions of a decision node can become more time consuming. But a well-designed abstract decision plan minimises the extent to which the time requirements grow with increasing number of covered examples.

The nature of decision planning knowledge representation methodology is such that it does not offer any means for a strong update. System knowledge base can be either directly supplied with inference knowledge specified by user or step-by-step induced from the field theory using Explanation Based Generalisation (EBG) technique which cares for weak updates. The lifecycle is then a sequence of weak updates. To ensure functionality of a strong update (Fig. 2), there have been implemented some processes of decision space filtering, aimed at reducing the number of decision nodes and the branching factor. We distinguish between following two filters:

- CONJUNCTIVE FILTER clusters decision nodes with the same *effect;* they get unified through conjunction of corresponding preconditions.
- FRAME FILTER checks relevancy of each of preconditions in order to avoid the frame problem ([1]). The algorithm has to be provided with the entire possible range of values each attribute can take. Possible clusters of decision nodes having the same effect and precondition describing entire discourse are eliminated.

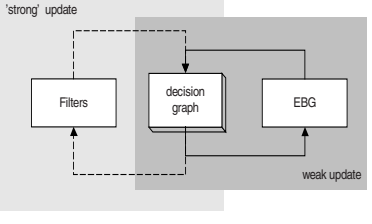

Figure 2 – Decision planning knowledge evolution lifecycle.

Time needed to learn a single example depends on the structure of the actual decision space (compared to the example to be accepted) and on the amount of the nodes to be parsed. The first objective makes an example to be learnt more difficult in the beginning of the learning process and the other objective makes it more time consuming with increasing number of accepted examples. Apart from the first 30 examples the time needed for learning is linearly proportional to the size of the decision space for our data set. When filtering the decision space the nodes have to be compared one to another (to put it simpler), thus the complexity is almost proportional to the square of number of nodes. The space of decision nodes does not grow with increasing number of positive examples proportionally. It saturates at certain point and consequently filtering takes just certain time (seconds) but does not grow up

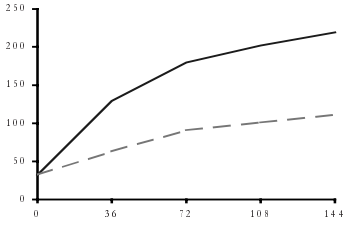

Figure 3 – Positive accepted examples/ number of nodes in the decision space

to infinity. Experiments showed that the best practice in maintaining the knowledge base consistent with respect to frequency of updates corresponds to the rhythm in which object-level data come. As each new piece of product has got usually about 30 new variants (4 – 6 new attributes) we were filtering the knowledge base after each 32 arrivals of new object-level data. This mechanism provides the knowledge base with filtered data ready for consultation whereas it minimises the time needed for knowledge induction. The solid line in the Fig. 3 gives the size of the decision space after several tens of weak update iterations (in terms of number of decision nodes). The effect of proposed filtering can be seen on the dashed line in the Fig. 3.

3 Conclusion and Future Work

The issue of knowledge maintenance is of immense importance throughout entire lifecycle of a knowledge-based system. Since general methodology for updating knowledge is missing, we have compared two machine learning methodologies for addressing this issue – *Inductive Logic Programming* (ILP) and *Explanation Based Generalisation* (EBG) within the framework of *Decision Planning* (DP).

Both approaches differ in number of aspects. The ILP approach generates a logic program, which can be directly used by the reasoning mechanism that mimics the expert's configuration process using standard methods of logic programming. In the case of DP the configuration mechanism is more elaborate. Graphs of interrelated decision nodes (decision graphs) on various levels of specificity are parsed, each representing either an action or another lower level graph.

Originally, we have suggested the ILP approach as an alternative for intelligent maintenance of knowledge base through a *strong* update, which corresponds to ILP analysis of all positive examples. Later, the w*eak* update proved to be simple to achieve due to the used knowledge representation: the new positive example is added as an exception in front of the system's base of rules. This takes just one single step and the system using the rule base after a sequence of weak update steps exhibits acceptable decrease of efficiency (linear function with respect to number of consecutive weak updates). Moreover, we have found a simple method (*Assertion 1*) how to estimate frequency of strong updates in dependence on the requested efficiency of our target KB system.

On the contrary, the central point of the decision planning approach is the weak update achieved by EBG. EBG updates the decision graph so that a new example is accepted. As this example is incorporated within already existing decision space structure, the extent to which its complexity increases is minimised. The time needed for the weak update here is considerably bigger than in the case of ILP approach. Decision planning implements the strong update by means of filters that reorganise the decision graphs in order to maintain efficiency of knowledge representation.

As decision planning requires strategic inference knowledge to be acquired from the user in order to formalise initial decision graph; its utilisation is envisaged mainly in areas where human expertise is available. The DP methodology just ensures the decision graph to be consistent with the evolving task domains. Our experience

verified that ILP is good at digging inference knowledge from the domain data and it is well suited even for areas with no or little of pre-specified human expertise. Efficiency of the resulting logic program can be estimated and used to find a proper balance between weak and strong updates. This makes ILP a robust and reliable mechanism for maintenance of domain knowledge throughout life cycle of the system.

Suggested properties of revision processes should be studied more carefully since they have to be taken into account whenever choosing an appropriate knowledge representation and knowledge extraction process for a dynamic application. Even here there have to be done decisions concerning the frequency with which the discovered theory has to be recomputed using a strong update instead of applying only a weak update of discovered theory. It is important to choose such frequency of strong updates, which suites best the dynamics of the system and which takes into account time and memory demands of the strong update. Finding balance between efficiency of weakly updated knowledge base and the result of time-consuming strong update belongs to the key problems within this area and it deserves development of rigorous methodology. Our case study shows that it is worth of considering ILP tools in this context. One of advantages they offer is that some useful upper estimates can be derived for the efficiency of the resulting programs.

Acknowledgements.

The research has been partially supported by the Esprit Project 20237 ILP2 and by INCO 977102 ILPNET2.

References

1. Davis E.: Representation of Common-sense Knowledge, Morgan Kaufmanns Publishers, Inc. California, 1990
2. Lavraè, N. et al.: ILPNET repositories on WWW: Inductive Logic Programming Systems, Datasets and Bibliography, *AI Communications*, Vol.9, No.4, 1996, pp. 157-206
3. Kodratoff, Y.: Machine Learning, In.: Hojat Adelli (ed.) Knowledge Engineering, McGraw-Hill Publishing Company, USA, 1990
4. Michalski R., Kaufmann K.: Data Mining and Knowledge Discovery: A Review of Issues and a Multistrategy Approach, In. Machine Learning and Data Mining: Methods and Applications (Michalski, Bratko, and Kubat Eds.) John Wiley & Sons Ltd, 1997
5. Pìchouèek, M.: Advanced Planning Techniques for Project Oriented Production, Ph.D. Dissertation, Prague, 1998
6. Pìchouèek M.: Meta-level structures for industrial decision making, In: *Proceedings of the Second International Conference of Practical Application of Knowledge Management*, London, April, 1999
7. Quinlan, J. R., Cameron-Jones, R. M.: Introduction of logic programs: FOIL and related systems, *New Generation Computing*, Special Issue on ILP, 13 (3-4), 287-312, 1995
8. Štìpánková O., Pìchouèek M., Mikšovský P.: ILP for Industrial Configuration, Tech. Rep. of Gerstner Laboratory - GL-74/98
9. Tansley D.S.W. and Hayball C.C.: Knowledge-based Systems Analysis and Design, Prentice-Hall, 1993
10. Wrobel,S., Wettschereck,D., Sommer,E., Emde,W.: Extensibility in Data Mining Systems, In: *Proceedings of the 2nd International Conference on KDD*, Menlo Park, CA, USA, August 1996, pp. 214-219