On Efficiency of Learning: A Framework

Jindřich Bůcha
Institute of Information Theory and Automation
Pod vodárenskou věží 4, Prague
bucha@utia.cas.cz

Abstract

The efficiency of learning is approached in the context of a problem solver. The problem is solved as a problem of integration of all basic cognitive functions. The feasibility of the approach is demonstrated on examples from a chess environment.

1. Introduction

This paper addresses machine learning (learning) [6]. In a more complicated environment, requirements on learning lead directly to the learning efficiency. Our hypothesis and/or assumption is that to solve the problem of efficient learning it is necessary and possible to solve it in a wider context, to solve it as a problem of efficiency of problem solver (PS). By PS we understand a system integrating all basic cognitive functions, i.e. identification, planning, implementation, learning, self-reflection, initialization and utilization of knowledge base (KB) functions. The justification for this hypothesis is that these basic cognitive functions could and should support each other to cover the requirements. (I do not know another way to cover the requirements.) We are going to test the hypothesis. Up to now, our analysis, experiments [2, 1] and hand-simulation has shown the feasibility of the approach. The paper describes the corresponding integration framework. The framework is presented on a general and conceptual level. There are many features characterizing our approach. At the beginning, I want to stress two of them. Two of the most important features enabling integration are an object-orientation and self-reflection.

Contemporary learning does not deal with complex environments, too much. For example, on ECML 97 [7] practically no paper presented learning used in really complex environment, e.g. in real world. Current applications of (inductive) learning concentrate on the utilization of single techniques [12]. Similarly, no ECML 97 paper presented learning in a more integrated system. The object-orientation is very frequent in AI today, but on ECML 97, I counted only 16% of learning papers dealing with this topic. It is not much. However, several systems are and were more advanced and ambitious [8]. The most advanced contemporary systems seem to be PRODIGY [11] and Soar [9]. These systems have also already showed that the integration of several cognitive functions is possible, that it is useful and efficient. They are incrementally extending their systems. I would characterize the approach of the corresponding projects as bottom-up.

My proposition is to start from the other end, i.e. try to apply top-down approach, try thorough design and only then implement and experiment. I would characterize this approach as use-case (i.e. requirement) driven, architecture centric, and iterative and incremental. I borrowed this label from the specification of UML (Unified Modeling Language) [10] that is a standardized language for object-oriented analysis and design. The presented approach could also be characterized as the object-oriented analysis and design of learning. UML is used in our approach and in the paper, too.

In this paper, I have no space to follow the detailed systematic PS design. I can refer about some of its current results, only. The paper has three main parts. In the first, an overall PS context is presented. Here, some of currently roughly 90 requirements on PS and learning are mentioned. In the second part, PS, its functions and relations, i.e. data model of PS, are described mainly by graphical means. Some properties of results of integration are described in the third part. I am going to illustrate the internals of PS on its hand-simulated work in chess environment, denoted EX1. In EX1, PS starts with about ten concepts, i.e. descriptions of object classes like Board, Figure, Field, Color, Opponent, and Teacher. Some are very trivial. PS can learn and use other more sophisticated concepts like fields relations, like Next, Neighboring, Previous, Row, Column, Diagonal, their Ordering, Covering, Figure Behavior, Configuration Behavior, Opening, Gambit, Occupied, Freed Field, Exchange, Defended Figure. This is our illustration of the learning efficiency.

2. Problem Solver Context

In this part, I am presenting basic assumptions about PS context. They have the form of five principles. For PS design, they represent requirements that must be fulfilled.
The first principle: Environment is a network of many dynamic objects. (Object and network are primitive concepts of the framework.).
This principle has important consequences: Any subnetwork of the network can be considered an object. Objects are composed from objects, again. There is a tremendous variability and ambiguity in ways in which Environment and/or object can be decomposed into lower level objects and in which set of objects can be composed into higher level objects. The objects can influence one another. They have input and output components. A concept of message can be used, i.e., we can use a mechanisms, in which a sending object sends a transmitted object(s) to a receiving object(s).
EX1: Chess Environment has the following objects: i) Board with Figures, with its State (Output) specifying a) PS Color and b) Side (Color) to Move, c) Final Result at the end of a game, d) Beginning of a game, ii) Teacher, iii) Opponent. The other objects are composed. They are e.g. Row, Column, Diagonal, their Ordering, Covering, Figure Behavior, Configuration Behavior, Opening, Gambit, Occupied, Freed Field, Exchanged, and Defended Figure.
The second principle: PS is an object of Environment. PS has a pre-specified goal. PS can control Environment in accordance with its goal. PS control is accomplished by means of its (basic) functions and by means of knowledge used by these functions. Knowledge is stored in knowledge base (KB). Planning is a PS function, which goal is to produce plan how to reach a PS goal. Implementation is a PS function, which goal is to implement plan by means of outputs applied in Environment and by using inputs from Environment. Learning is a PS function, which goal is to produce knowledge.
EX1: PS has input components: 64 components indicating field occupation, e.g. OccupiedBy(a2,WP), that means field a2 is occupied by figure white (W) pawn (P), component synchronizing a game that has 4 sub-components, i.e. PS-Color, specifying a color PS is playing with, SideToMove specifying which color is to move, FinalResult specifying a game result and Beginning, specifying beginning of a game, component specifying Teacher Evaluation, it can have values ! or ? (good or bad). PS has three output components: component MakeMove, specifying that PS is to move, component specifying where to move from, e.g. From(a2) and component specifying where to move to, e.g. To(a3), that reads move from field a2 to field a3.
The first two principles have important consequences: The whole object-oriented approach can be applied to PS and its parts, e.g. to KB. PS can control itself (self-reflexivity). It means, PS functions, e.g. planning, learning, can have recursive character. PS can have meta-knowledge. Knowledge can play a role of both knowledge and data. Knowledge has a role of data when it is an object for recursive PS. Knowledge and PS is more homogenous, i.e. PS does not need different kinds of knowledge for external Environment and for itself, e.g., for PS plans, objects.
The third principle: Knowledge is an approximate description (of the behavior) of an object.
These principles have again consequences: The more accurate goal of learning is to improve a precision of knowledge. Learning is an iterative process. It should be possible to describe all kinds of objects of Environment, as the objects of Environment are interconnected and the impossibility to describe one part may cause the impossibility to describe the whole Environment. Then knowledge should be general one, e.g., able to describe definite and/or stochastic, static and/or dynamic, simple and/or structured objects. Learnt object, which PS has already knowledge about, can be used as a tool for PS, e.g., tool for next learning (see later).
The fourth principle: Environment is relatively very stable, i.e. a very little part of Environment is changing during PS work.
The fifth principle: PS (learning) should cope with its complexity.
Some consequences of the previous principles and corresponding requirements are already oriented on the solution of complexity. There are additional possibilities, e.g. induction, focus of attention concentrated on changes, focus of attention concentrated on identified inconsistencies in descriptions, imitation, i.e., overtaking knowledge created by other members of society of similar PS.

3. Problem Solver

In this part, there is a rough proposal how to implement all identified requirements. It shows relations among basic KB structures and basic PS functions. I use graphical language UML (Unified Modeling Language) in presented Figures, see e.g. Fowler [3]. The short explanation of used language symbols follows: Named rectangle, with two additional lines inside, means a class, e.g. Learning. The class describes its objects. Named man icon means an active (external) class, e.g. Environment. (Named) line connecting two classes means an association relation, e.g. Input. It is used for a data flow, event and other relations. (Named) arrow with a diamond head means that a class is a part of another class, e.g. IOS is part of Description. (Named) arrow with a triangle head means a class-subclass

relation; e.g. Learning is a subclass of Basic Function. For the description, I use the names of classes corresponding to design entities. The class names are referenced with capital letters.

The core of Knowledge Base (see Figure 1) is De-

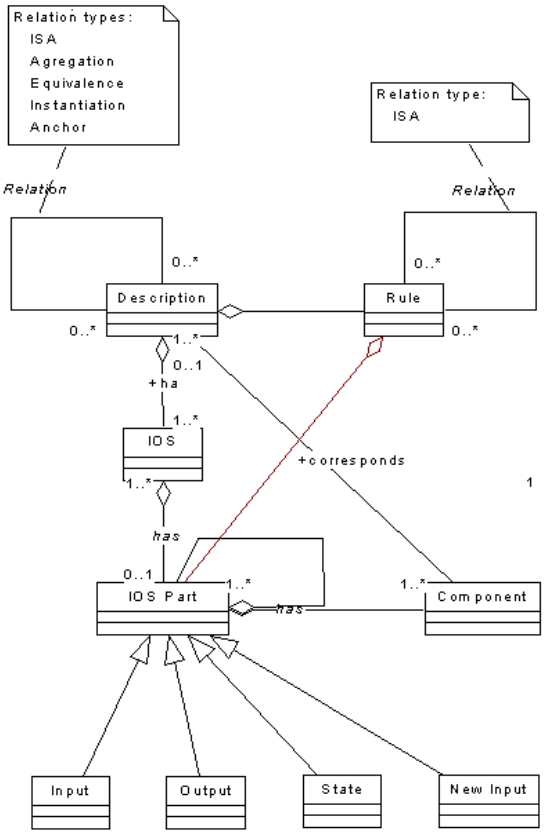

Figure 1: Knowledge Model

scription. This is describing a class of objects. This can be related to other Descriptions via various kinds of relations, e.g. ISA (generalization), aggregation, association, instantiation, and anchor relations. Description has Rules. Rule can be generalized, i.e. related to other Rule by ISA relation. Description has IOS (Input, Output and State) part. This is divided into four parts: Input, Output, State and Next Input. Each IOS part has Components. For each Component, there is a corresponding Description specifying a range of its values. For each of these entities, there is a corresponding Instantiated entity, when needed. For example, there can be Instantiated Descriptions corresponding to Description, Instantiated Components corresponding to Component. Next Input is an alternative to Output; Output has character of action, Next Input has character of state. EX1: Input is initial Board situation, Next Input is a Board situation after the move.

PS (see Figures 2 and 3) is connected with Environment via its Input and Output (I/O). Outputs are produced by Implementation function. Im-

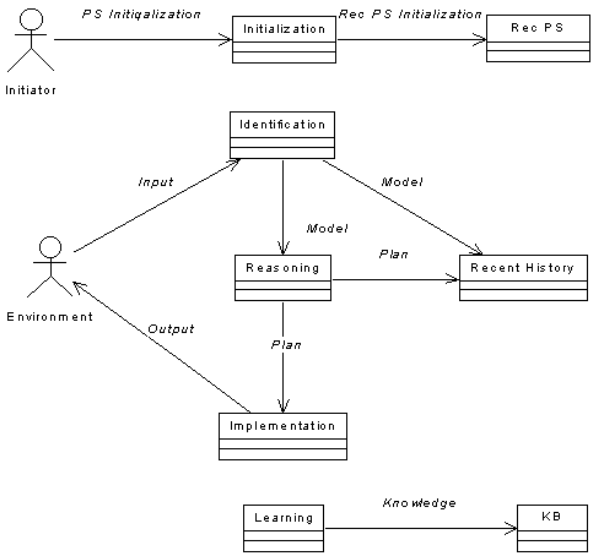

Figure 2: Special Relations of PS Functions

plementation is based on Plan, prepared by Planning function and on Input provided by Environment. I/O is stacked in Recent History. All Recent History is input for Identification, Planning and Learning functions. With this, Identification produces Model. Planning uses Model to produce Plan. Learning produces Knowledge. All functions use KB and are controlled by Recursive PS. I/O, Plan, Model has character of Instantiated Description, Knowledge has a character of Description. These relations are illustrated on figures. According to UML methodology, there are specific and common views of PS, there. A more detailed function description follows.

Identification tries to find in KB Description and

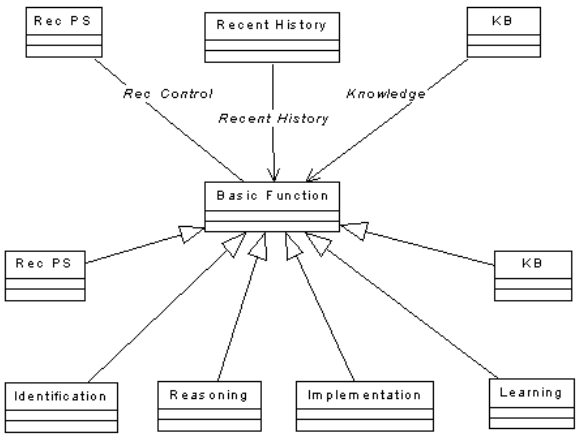

Figure 3: Common Relations of PS Functions

instantiation of this Description that corresponds to the I/O behavior of Environment, as gathered in Recent History. It means it tries to find Model of

Environment. As a part of Model, Problem is identified, too. Later, we will see that Identification plays a role of an intelligent PS input. Planning tries to find Plan that transmits Environment represented by its Model from current Problem (state) to the Goal (state). It can create subproblems. Implementation implements prepared Plan. It means, it produces Outputs, applied in Environment according to Inputs from Environment, Plan and Plan state. Later, we will see that Implementation plays a role of an intelligent PS output. Knowledge base functions saves and retrieves knowledge. Learning creates Description using Recent History as a source of training set and using identified Problem as evaluation of this training set. Self-control is called RecPS, i.e. recursive PS. It works in a similar way, like PS. It will be implemented as recursive PS. RecPS Environment is the internals of PS. RecPS I/Os are PS I/Os and characteristics of PS basic functions' Inputs, Outputs and State, e.g. success. The level of recursion is restricted by necessity to initialize the work of RecPS. Problem for RecPS is a not-successful work of some basic function. The success can be immediate or long-term. For example, it can be announced immediately that Learning was not able to create Description from given training set. And, it can take some time to recognize that some Description is wrong. Initialization initiates KB with built-in knowledge. In range of this, it initializes connection between KB and PS I/O Components and in similar way, it initializes connection between KB and RecPS I/O Components. It initializes the work of RecPS, too.

In a simple Environment, to identify (the only one) object and its state need not be a problem, to plan solution in this Environment, described as a simple object, need not be a problem, to learn Description of this object need not be a problem and to implement Plan need not be a problem. Recursive control will not be necessary and KB will be trivial. I call the corresponding PS functions simple PS cognitive functions. However, the real Environment is different.

EX1: Initial KB (see Figure 4) contains classes (object descriptions) (Initial) Plan, Environment, Field, Figure, Color, PS, Opponent. (I am using Plan here with meaning of Description, not with meaning of Instantiated Description, as above.) Some of them are very simple, they have no IOS, they may (initially) only exist (Figure, Color, PS, Opponent). From the graph theory viewpoint, they are leaves of KB graph. These descriptions have the corresponding I/O components described above. Input of Plan is Output of Environment and vice-versa. PS begins practically without any knowledge. Its moves are selected randomly. There is not too much chance for successful move.

To implement (full) cognitive functions it is necessary to extend the existing basic functions in such a

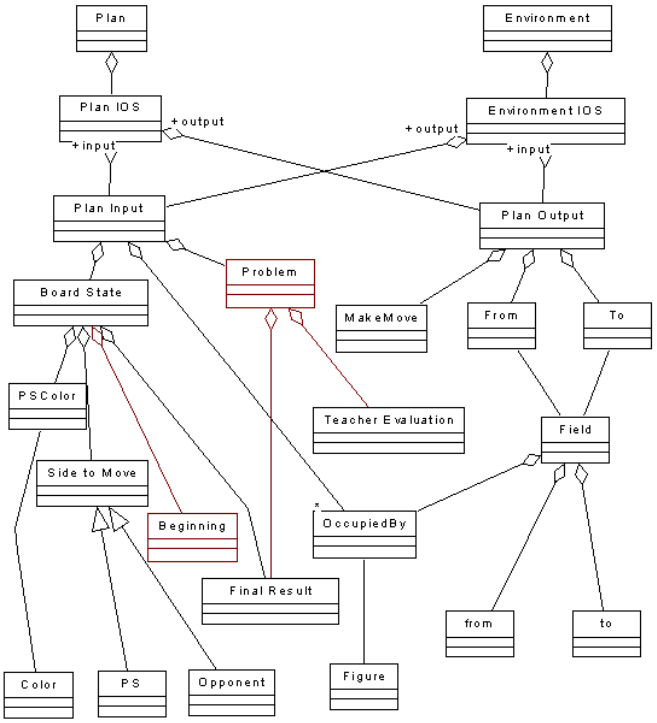

Figure 4: Initial KB

way to cover the work with both simple and object-oriented Environment, with both deterministic and stochastic objects, with both static and dynamic objects etc. To solve the object-oriented character of the Environment the concept of using known object as a tool will be used: PS will consider the tool to be a part of its Input or Output or both. In this way, PS will shift its border with the Environment.

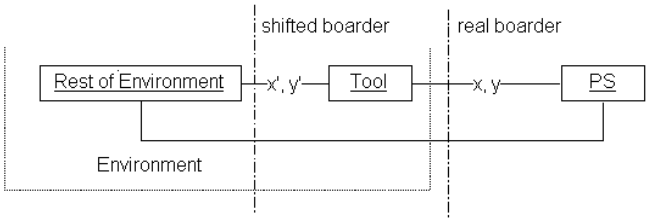

Figure 5: Shifted Boarder

describe it, little bit more (see Figure 5). Let us have PS, its Environment, some Tool (as an object of Environment) and Rest of Environment. PS has Inputs to perceive Environment, the part of them is x. PS has Outputs to influence Environment, and the part of them is y. Together, x and y form the part of the real border between PS and Environment, i.e. border between PS and Tool. Let us assume that PS can 1) learn characteristics of Tool in such a way that it can 2) recognize Tool, if it knows a sequence of values of x and y, 3) specify a sequence of values of x' and y' (which form the border be-

tween Tool and Rest of Environment, and which are not accessible to PS). x' and y' are called modeled Inputs and Outputs. The equality of modeled Inputs and Outputs, both in Environment and in PS, can be interpreted as a shift of boarder between Environment and PS. Then PS can control Environment in such a way as if it has a shifted boarder with Environment, i.e. the boarder partially composed from x' and y', instead of x and y. From the assumption that PS can learn and use tools directly connected to it follows, that it can learn also tools not directly connected. In addition, it means that input information for learning, i.e. Outputs and Inputs, can be iteratively extended by information about modeled Inputs and Outputs. One of tools can be a teacher. One of teachers can be a society. PS can use knowledge created by society of similar PSs, i.e. learning can have a distributed character. Some of tools can be tools for communication with similar PSs. One of tools can be an approach to creation and/or learning. That tool can be used to design learning itself.

4. Properties of integrated PS

Here, I am going to mention several examples of PS work enabled by PS integration.
In a similar way, in which partial Plan descriptions are created, PS can create Descriptions of behavior of Board and its parts, i.e. partial Descriptions of Environment. On evaluated examples, teacher can demonstrate possible behaviors and PS can observe this behavior and generalize it. Based on Input information about Board situation and about changes of the situation, PS can create Descriptions in a very similar way as it can create Descriptions of its Plans: PS is oriented on change. By this way, it gets information about Field from which a move was made and to which it was made.
RecPS observes the work of Learning with the goal to help, e.g. to create a plan how the create Descriptions directly. Let us assume, PS can learn a (partial) Plan WPa2-a3. Moving white pawn from field a2 to a3. Let us assume that it observes Learning creating two Descriptions, partial Plans WPa2-a3 and similar WPb3-b4. RecPS observes parts of these Descriptions as meta-components indicating existence of Description, IOS, Input, Output ... By comparing these two meta-training examples, it can be found that they differ in values of Fields. Description that is more general can be obtained by generalizing different parts of Description. Interpreted pre-built RecPS plan for Learning instructs (basic) Learning via its Output meta-components how to create a new Description, PSPMove. This is a basis of analogy. Notice that meta-components have a dynamic character in a sense, that sets of I/O components are not fixed, not pre-built.

Plan PSPMove specifies a relation between two

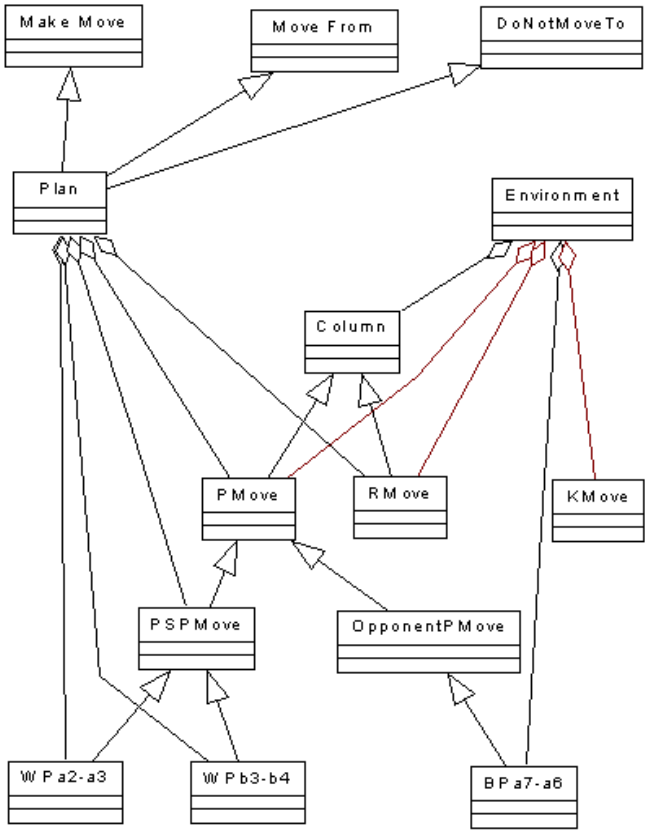

Figure 6: KB Snapshot

objects. Plan OpponentPMove, created in a similar way, uses (nearly) the same relation. Using above described analogy, a new Descriptions can be created, PMove etc. (see Figure 6).
PS functions are implemented in shifted border context: To identify object from Rest of Environment it is necessary to identify tool, T and to use T as a tool to get x', i.e. use Implementation to apply y, and Meta-Implementation to derive x'. To learn object from Rest of Environment it is necessary to identify tool, T, to use T as a tool to get x' and apply y', i.e. use Meta-Implementation to derive x', and Implementation to apply y and indirectly via tool y' and learn description on level of x' and y'. In accordance with the tool concept, newly created Description extend the set of components of I/O. This together with use of KB and recursive control of Identification and Implementation form a basis of intelligent I/O.

5. Conclusion

The presented approach showed feasible in a sense that there has not been found any substantial obstacles impeding the implementation of the framework.

It is necessary to continue in the PS design. Our proposition is to continue with a detailed analysis and design of basic functions and KB using contemporary software engineering approach, i.e. using object-oriented CASE (computer aided software engineering) tool.
In higher cognitive processes, self-reflection plays an important role. AI is a cognitive process of stepwise understanding the phenomenon of intelligence. What is the state of self-reflection in AI? According to my evaluation - generalization, AI is just on the edge of its self-reflection. What do I mean by this? I mean that there are parts of AI community, but only small parts now that are just beginning to use specific benefits of AI work, i.e. they are beginning to interpret the results of AI process to control AI process itself. I tried to use this knowledge already here.

6. The references

References

[1] J. Bůcha, "Incremental Learning", *Computers and Artificial Intelligence* , vol. 4, no. 4, pp. 325-334, 1985.

[2] J. Bůcha, *Problem Solving with Knowledge Represented by Probabilistic Automata Network*, Ph.D. Thesis, Czechoslovakian Academy of Science, Prague, p. 110, 1978 (in Czech).

[3] M. Fowler and K. Scott, *UML Distilled*, Addison-Wesley, N.Y., USA, 1997.

[4] I. M. Havel, "The Concept of Indirectness in Artificial Intelligence", *Kybernetika*, vol. 8, no. 2, pp. 154-164, 1972.

[5] R. S. Michalski, J. G. Carbonell and T. M. Mitchell, *Machine Learning, An Artificial Intelligence Approach, Volume II, Morgan Kaufmann Pub., Los Altos, CA, p.738, 1986.*

[6] R. S. Michalski, "Understanding the Nature of Learning: Issues and Research Directions", in [5], pp. 3-25, 1986.

[7] M. van Someren and G. Widmer (eds.), Machine Learning-97, Proceedings of 9th European Conference on Machine Learning*, Prague, Czech Republic, 1997.*

[8] "Special Issue on Knowledge Representation and Reasoning Systems", SIGART Bulletin*, vol. 2, no. 3, 1991.*

[9] The Soar Project at Carnegie Mellon University*, http://www.cs.cmu.edu/afs/cs.cmu.edu/project /soar/public/ www/home-page.html, 1998.*

[10] UML Summary*, Rational Software Corporation, http://www.rational.com/uml/, September 1997.*

[11] M. Veloso et al ., "Integrating Planning and Learning: The PRODIGY Architecture", J. of Experimental and Theoretical Artificial Intelligence*, vol. 7, no. 1, 1995.*

[12] F. Verdenius and M. van Someren, "Applications of inductive learning techniques: a survey in the Netherlands", AI Communications*, vol. 10, no. 1, 1997.*