CN2-MCI: A two-step method for constructive induction

Stefan Kramer
di_stefan@ai.univie.ac.at
Austrian Research Institute for
Artificial Intelligence
Schottengasse 3
A-1010 Vienna
Austria

March 8, 1994

Abstract

Methods for constructive induction perform an automatic transformation of description spaces if representational shortcomings deteriorate the quality of learning. In the context of concept learning and propositional representation languages, feature construction algorithms have been developed in order to improve the accuracy and to decrease the complexity of hypotheses. Particularly, so-called hypothesis-driven constructive induction (HCI) algorithms construct new attributes based upon the analysis of induced hypotheses. Well-known HCI-systems analyze decision trees, or employ a coarse-grained analysis of decision rules. This paper introduces a new constructive operator o_κ and documents its applicability in the usual HCI-framework. o_κ uses a cluster algorithm to map selected features into a new binary feature.

A new method for constructive induction, CN2-MCI, is described that applies o_κ as its only constructive operator to achieve a *fine-grained analysis of decision rules*. The output of CN2-MCI is an inductive hypothesis expressed in terms of the transformed representation, given training examples as input. CN2-MCI is theoretically and empirically compared with existing methods for constructive induction.

Contents

1 Introduction

Constructive Induction is the subject of a longstanding and still vital research effort to overcome representational shortcomings that deteriorate the quality of learning. If the initially given representation yields poor learning results, constructive induction methods perform an automatic, problem-oriented transformation of representation spaces to facilitate learning.

In the context of concept learning and propositional representation languages, constructive induction methods have been developed which construct new attributes out of existing attributes. These algorithms employ feature construction in order to improve the accuracy and to decrease the complexity of hypotheses.

Particularly, a number of methods have been proposed over the last years that construct new attributes based upon the analysis of induced hypotheses. These so-called methods of hypothesis-driven constructive induction (HCI) typically construct new attributes in iterations, where each iteration involves a learning step and a step which constructs new attributes based on the hypotheses of the learning step.

CN2-MCI is a new method for constructive induction, which works in a similar way as many HCI-systems. CN2 [Clark & Niblett, 1989; Clark & Boswell, 1991], a well-known selective induction algorithm, provides the induced hypotheses that are analyzed by other components of CN2-MCI. CN2-MCI iteratively performs feature construction by means of a new, general constructive operator o_K. However, CN2-MCI differs significantly from HCI-systems like FRINGE [Pagallo, 1989; Pagallo & Haussler, 1990] and AQ17-HCI [Wnek & Michalski, 1992a+b] in that it processes training examples in addition to induced hypotheses. Therefore, it can be categorized as a multi-strategy constructive induction system (MCI).

1.1 Matheus' framework for feature construction

CN2-MCI fits well into a framework for feature construction which was introduced by Matheus [Matheus & Rendell, 1989; Matheus, 1991]. The following terms will be used in the description of CN2-MCI:

- A *constructive operator* is defined as a function mapping a tuple of existing features into a new feature.
- A *constructive operand* is a tuple of features to which a constructive operator is applied.
- A *constructor* is defined as a operator/operand pair, until it becomes a new feature.

Furthermore, Matheus distinguishes four inherent aspects to feature construction. They are not to be understood as phases of the feature construction process, although many existing systems including CN2-MCI perform phases corresponding to these aspects.

1. *Detection:* Feature construction is only required if the original feature set is insufficient for the selective induction algorithm to acquire the target concept. Wnek and Michalski [1992b] state the reason for the importance of detection as follows:

   > "In a domain description, irrelevant attributes play the same role as noisy examples in training data. In both cases, removing noise from the training set speeds up and improves induction."

   Briefly, the construction of irrelevant features should be avoided, because irrelevant features deteriorate the quality of the induced hypotheses. Another reason for the relevance of detection is the constructive process itself, which might be computationally expensive.

   Matheus and Rendell [1991] enumerate two approaches to detecting when a transformation of the representation spaces is required. One possibility is to decide on the basis of the initial data set. Another method is to analyze a concept description in order to detect the need for constructive induction. Another approach to detection is taken by CN2-MCI: The system basically performs a kind of look-ahead: it applies its constructive operator and evaluates its results. If the quality of the hypothesis induced in the transformed representation is worse than the original hypothesis, CN2-MCI concludes there is no need for a representation change.

   Evidently, there is a close connection between detection and another aspect of feature construction, namely the *evaluation* of new features. The *evaluation* of new features and the aspect of detection are both essentially instances of the more general problem of how to evaluate a given representation.

2. *Selection:* Since the set of constructors is potentially very large and its detailed evaluation is intractable in general, there needs to be a selection of a subset of constructors which are to be turned into actual features. The number of constructors is determined by the number of constructive operators and the number of existing attributes. The selection can be further divided into the following sub-steps:

   (a) *Selection of constructive operands:* In the first step, the existing features from which the new features are to be constructed are selected. Step one reduces the number of potentially constructed features.

(b) *Application of the constructive operators and final selection* In the second step, the constructive operators map the constructive operands into the new features. Based on the results of the applied operators, the final selection of constructors can be made.

3. *Generalization:* Generalization of definitions might effect an improvement, if the new features are too specific to capture a meaningful relation between variables. However, most of the existing systems do not perform a generalization of selected constructors.

4. *Evaluation:* The evaluation of new features is necessary for the same reasons as detection. The goal of an evaluation is to find a subset of the new features which should actually be included in the description of the training examples. Another important aspect of evaluation is the need for a global stopping criterion, which is rarely described explicitly in the literature known to the author. Pragmatically, CN2-MCI takes the same approach as the one to detection to determine when to stop the constructive iterations.

1.2 A taxonomy of constructive induction systems

Wnek & Michalski [1992b] introduce a taxonomy of constructive induction systems. This taxonomy is useful for the classification of the presented system.

- *Data-driven constructive induction (DCI):* DCI analyzes the training examples in order to perform constructive induction. Specifically, new descriptors are found by the search for interrelationships among examples, attributes and concepts.

- *Hypothesis-driven constructive induction (HCI):* HCI refers to methods of transforming representation spaces by analyzing generated inductive hypotheses.

  Methods for hypothesis-driven constructive induction typically construct new attributes in iterations, where each iteration involves a learning step and a step which constructs new attributes based on the hypotheses of the learning step. The rationale behind this approach is to view the hypothesis as information already mirroring the relation between the learning capabilities and the problem representation. This information is utilized to guide the construction of new attributes in order to provide the learner with a transformed representation which is more suited to its learning capabilities. In particular, these methods provide a way to encode relations between variables as binary attributes.

- *Knowledge-driven constructive induction (KCI):* These systems apply expert-provided domain knowledge to construct new descriptors. Furthermore, representation changes can be validated by a domain expert.

- *Multi-strategy constructive induction (MCI):* MCI-systems combine different approaches to the transformation of description spaces. In this taxonomy, a system utilizing inductive hypotheses and input data for constructive induction is categorized as MCI.

At last, the name CN2-MCI can be explained: The first step of the *selection*, namely the *selection of the constructive operands*, is based on the analysis of the *hypothesis*, whereas the second step, the *application of the constructive operator*, requires access to the *training examples*. The selective learning algorithm CN2 [Clark & Niblett, 1989; Clark & Boswell, 1991] provides the rules which are analyzed for detection, selection and evaluation.

Despite its categorization as MCI, it should be noted that especially the top-level of CN2-MCI works in a very similar way as systems performing HCI.

1.3 Motivation

Well-known methods for HCI are based on the analysis of decision trees or employ a coarse-grained analysis of decision rules. FRINGE [Pagallo, 1989; Pagallo & Haussler, 1990], for instance, utilizes repeatedly occurring subtrees at the "fringe" of a decision tree as definitions of new features. Since the same procedure is hardly applicable to decision rules, AQ17-HCI [Wnek & Michalski, 1992a+b] uses the most relevant disjuncts of a hypothesis as the definition of a new attribute. However, little attention has been paid to the possibility of a more fine-grained analysis of decision rules. There are two closely related major motivations for this research: On the one hand the goal was to generalize the FRINGE idea in order to make it applicable to decision rules. On the other hand, it appeared promising to take into account the structure of rules, which are composed of attributes and their respective values. The construction process has the same input as AQ17-HCI, namely decision rules, but goes further into detail of a rule in the course of the construction process.
There are two expected benefits of a more fine-grained analysis of decision rules. Firstly, we expected a decreased complexity and an increased accuracy, similar to the effect of FRINGE-like feature construction. Secondly, we wanted to obtain more intelligible features than the ones constructed by AQ17-HCI. The constructed features should encode relations between features, which are "relevant" with respect to the class attribute. Unlike AQ17-HCI, the features involved should be clearly identifiable by the user.

2 CN2-MCI: Description of the Method

2.1 The top-level of CN2-MCI

The top-level of CN2-MCI works in a similar way as systems performing hypothesis-driven constructive induction (see Fig.1). The selective induction algorithm CN2 [Clark & Niblett, 1989; Clark & Boswell, 1991] provides the rules which are analyzed for detection, selection and evaluation. After a single, initial learning step, the system starts constructing attributes in iterations, one in each iteration. CN2-MCI constructs *one binary attribute out of two existing attributes in each iteration*. The approach to detection taken in CN2-MCI is to start constructing features in iterations and record the quality change of the induced hypothesis in each iteration. There exists a threshold τ which specifies for how many iterations the quality is allowed to get worse relative to the best iteration so far. If after τ iterations the best hypothesis so far is the one found in the initial representation, CN2-MCI concludes there is no need for a representation change. This approach to detection is useful in avoiding the construction of irrelevant features, but obviously does not take into account the second argument for detection, i.e. it could be computationally expensive.

After the construction of a new attribute, the examples are updated. The new description of the examples contains the additional constructed feature. Following another learning step, the quality of the representation change is estimated using the quality change of the hypothesis which is induced in the transformed hypothesis space. A lexicographic function tests if the current hypothesis is better than the best one so far, considering the change in accuracy and complexity. In order to determine the change in accuracy, the set of training examples has to be split into a primary set and a secondary set [Bloedorn et al., 1993]. The primary set is used for learning via CN2, and the secondary set is used for assessing the quality of the representation through the accuracy of the generated inductive hypothesis. The reason for this is the need for a seperation between learning and testing, which is a precondition of a reasonable evaluation of learning results. Testing the quality of a new representation has to be done independently of the testing examples that are employed for testing the final hypothesis of CN2-MCI.[1] In such a way, a stopping criterion can easily be obtained. Like the approach taken to detection, the stopping criterion of the algorithm is based on testing for an improvement of the current hypothesis relative to the best hypothesis so far. In fact, detection and the stopping criterion are the same mechanism in CN2-MCI. If there are τ iterations without improvement relative to the best hypothesis found so far, CN2-MCI stops with the respective, optimal representation and re-learns a final theory using *all training examples*, and not just the primary

[1] If the accuracy of the hypothesis is not explictly tested on a secondary training set, the whole training set can be used for learning. In [Pfahringer, 1994a], the MDL-heuristic [Rissanen, 1978] is utilized to avoid the need for a secondary training set.

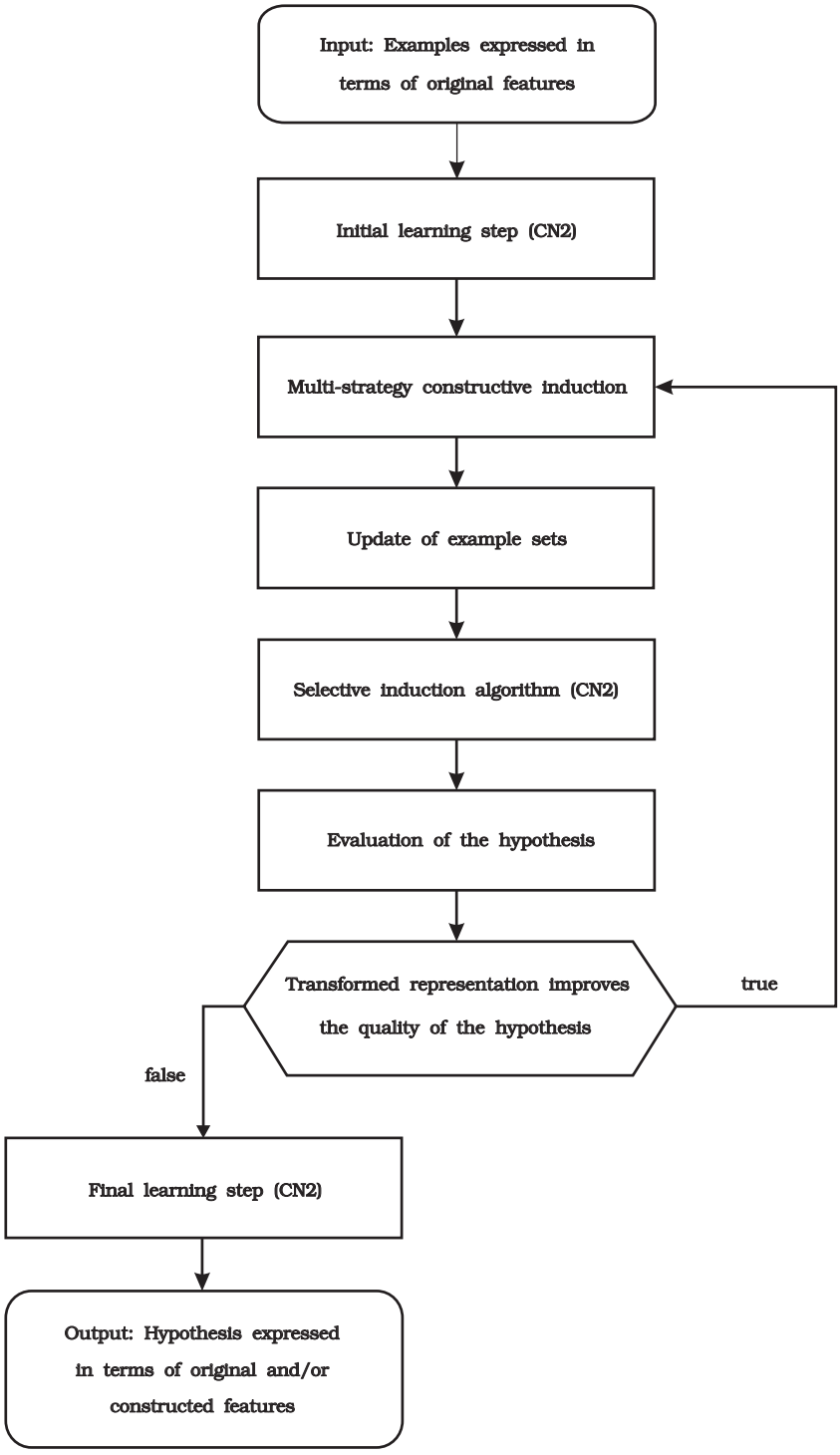

Figure 1: Schema of CN2-MCI

set of training examples.

Generally, CN2-MCI provides a simple solution to the problem of evaluation by applying representation changes and determining the *quality relative to the original representation*. In other words, the absolute quality of descriptors might be of interest to learning theory, but pragmatically CN2-MCI relates the originally given representation to the one it can achieve by applying its constructive operator.

2.2 Multi-strategy constructive induction in CN2-MCI

The construction process by means of the operator o_κ is part of the MCI module (see Fig. 1) and consists of two steps (Fig. 2). The first step includes the

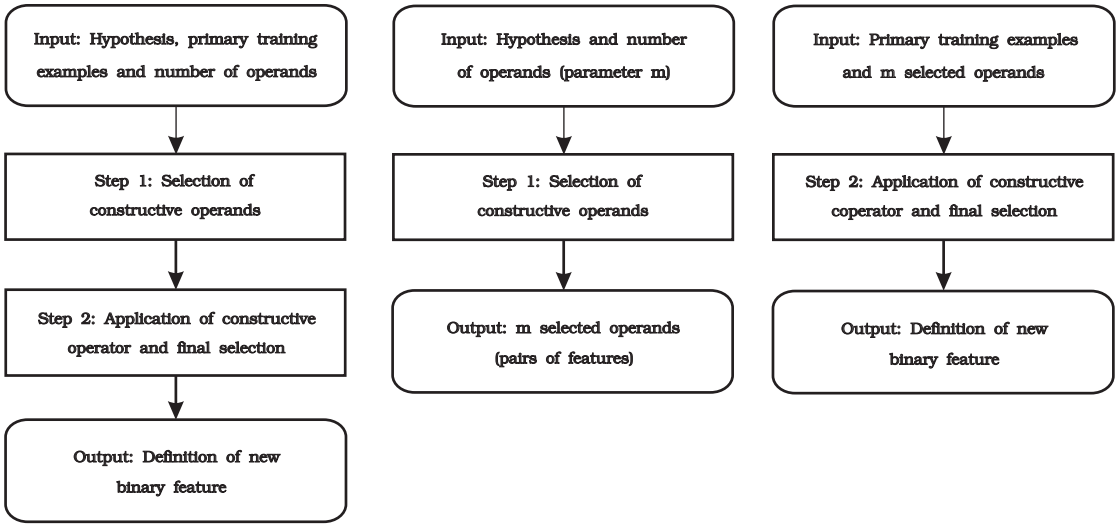

Figure 2: Two Steps of MCI in CN2-MCI

selection of the constructive operands, i.e. the pairs of attributes from which the new attribute is to be constructed. CN2-MCI selects m attribute pairs as constructive operands based upon the co-occurrence of the attributes in important rules of the hypothesis. The number of attribute pairs, m, is user-defined. In the second step, the constructive operator maps the respective two constructive operands into a new boolean feature. Following the application of o_κ to the m selected operands, a single constructor will be chosen as new feature. The final selection is based on the evaluation of the application of o_κ. In our approach, the constructive operator searches for a "good" partitioning of pairs of the values of the constructive operands into two clusters, each cluster corresponding to a value of the new boolean feature. After the applications of the constructive

operator, CN2-MCI selects the best constructor according to the best resulting partitioning.

If one is willing to put up with local optima, one way to obtain a partitioning is through a hierarchical-agglomerative cluster algorithm. This kind of algorithm merges those two clusters of pairs of values which result in the maximum improvement of some evaluation function. A (hill-climbing) cluster algorithm was chosen for CN2-MCI because an exhaustive search for an optimal partitioning is too expensive and not necessary to obtain good results.

There are m cluster analyses to be performed, because the best selection of constructive operands is unknown a priori. It is crucial to keep in mind that the optimal selection of the operands according to the selection heuristic does not necessarily lead to an optimal result of the cluster algorithm. Fortunately, a number of cluster steps can be saved if the m constructions are performed "in parallel" and a kind of A*-search is applied.

The selection of operands is presented in the next section. In the subsequent section, the cluster algorithm is described in detail. Finally, the use of A* is discussed to demonstrate how m "parallel" constructions (i.e. hierarchical agglomerative cluster algorithms) can be performed efficiently.

2.3 Step 1: Selection of constructive operands

The goal of the selection method is to encode the most relevant relations among existing attributes as binary attributes. The common occurrence of two attributes in many important rules is a necessary condition for a successful application of the constructive operator in the second step of the construction algorithm.

As mentioned above, CN2-MCI employs a heuristic S to select those pairs of attributes which occur in as many relevant rules as possible; in this stage the system abstracts from the actual values of the attributes.

Let $weight(r_i)$ be the number of all examples covered by rule r_i. Moreover, let us write $A_j \in r_i$ if attribute A_j occurs in rule r_i. CN2-MCI selects the pairs of attributes A_i, A_j, for which

$$S(A_i, A_j) = \sum_{A_i \in r_k \wedge A_j \in r_k} weight(r_k)$$

is maximal.

Example:

Consider the following rules induced by CN2 in a problem domain:

$r_1 : if\ (A_1 = t) \wedge (A_2 = t) \wedge (A_3 = f)\ then\ (class = +)$ [20,0]
$r_2 : if\ (A_1 = t) \wedge (A_2 = f)\ then\ (class = +)$ [10,0]
$r_3 : if\ (A_1 = f) \wedge (A_2 = f) \wedge (A_3 = f)\ then\ (class = -)$ [0,15]
$r_4 : if\ (A_2 = f) \wedge (A_3 = f)\ then\ (class = -)$ [0,5]
...

Attributes A_1, A_2 and A_3 do not occur in other rules of the hypothesis. The numbers in brackets denote the number of examples of a class covered by rule r_i. [20,0] means there are 20 positive examples covered by rule r_1 and no negative examples. The weight of the rule is computed as the sum of these two numbers. Then

$S(A_1, A_2) = weight(r_1) + weight(r_2) + weight(r_3) = 20 + 10 + 15 = 45$
$S(A_2, A_3) = weight(r_1) + weight(r_3) + weight(r_4) = 40$
$S(A_1, A_3) = weight(r_1) + weight(r_3) = 35$

According to the value of S, CN2-MCI prefers the first pair of features to the second, etc.

2.4 Step 2: Application of the constructive operator and final selection

2.4.1 The constructive operator o_κ

o_κ is a hierarchical agglomerative cluster algorithm [Eckes & Roßbach, 1980; Jain, 1986] that constructs a partitioning of the values of the operands selected in the first step of the algorithm. The initial clusters contain the single pairs of the Cartesian product of the operands. Subsequently, in each step of the algorithm those two clusters are merged which result in the smallest increase of the evaluation function.

Next, a function E is defined to evaluate a partitioning into clusters. A cluster consists of pairs of values of the selected operands. Each partitioning into n clusters corresponds to an extensional definition of an n-valued attribute. The function E evaluates the occurrences of the values of the attribute-pair in the training examples. The function considers the relevance (i.e. the number of examples) as well as the impurity of the attribute values relative to the class attribute. If the value of the function is large, the quality of the partitioning is bad, because it contains large, unbalanced clusters.

Let n be the number of clusters in a partitioning c. Function E is defined as follows:

$$E(c) = \sum_{i=1}^{n} \frac{p_i * n_i}{p_i + n_i},$$

where p_i is the number of positive instances from the training examples for which the projection to the two attributes is contained in cluster i of partitioning c. Analogously, n_i is the value for the negative examples.

The formula represents the weighted impurity, which is to be minimized in order to avoid large, impure clusters. Alternatively, any other function estimating the a goodness of split of decision trees could be used as evaluation function. It is crucial to consider that the operator performs hill-climbing and is therefore generally not able to find globally optimal solutions. In constrast to o_κ, an operator introduced in [Breiman et al., 1984] is capable of finding optimal solutions. There are two major differences to be noted here: Firstly, o_κ is applicable to any number of operands, and not just one. Secondly, "Breiman's operator" only finds globally optimal solutions, if the number of classes is *two*. o_κ is a feasible way to compute "good" partitionings given n classes. Obviously, non-hillclimbing search algorithms could be employed as well to find "good" or optimal partitionings.

Example:

Two attributes, A_1 and A_2 are selected as operands (see Fig. 3). The attributes can take the following values: $Dom(A_1) = \{t, f\}$ and $Dom(A_2) = \{t, f\}$. The initial clusters are $c_1 = \{\{< t, t >\}, \{< t, f >\}, \{< f, t >\}, \{< f, f >\}\}$ and are evaluated by
$E(c_1) = \frac{23*2}{23+2} + \frac{12*3}{12+3} + \frac{1*1}{1+1} + \frac{2*16}{2+16} = 6.51.$

The numbers in brackets in Fig. 3 denote the class frequencies of the selected attributes taking the respective values in the clusters. For instance, there are 23 positive instances where $A_1 = t$ and $A_2 = t$, and 2 such negative instances, etc.

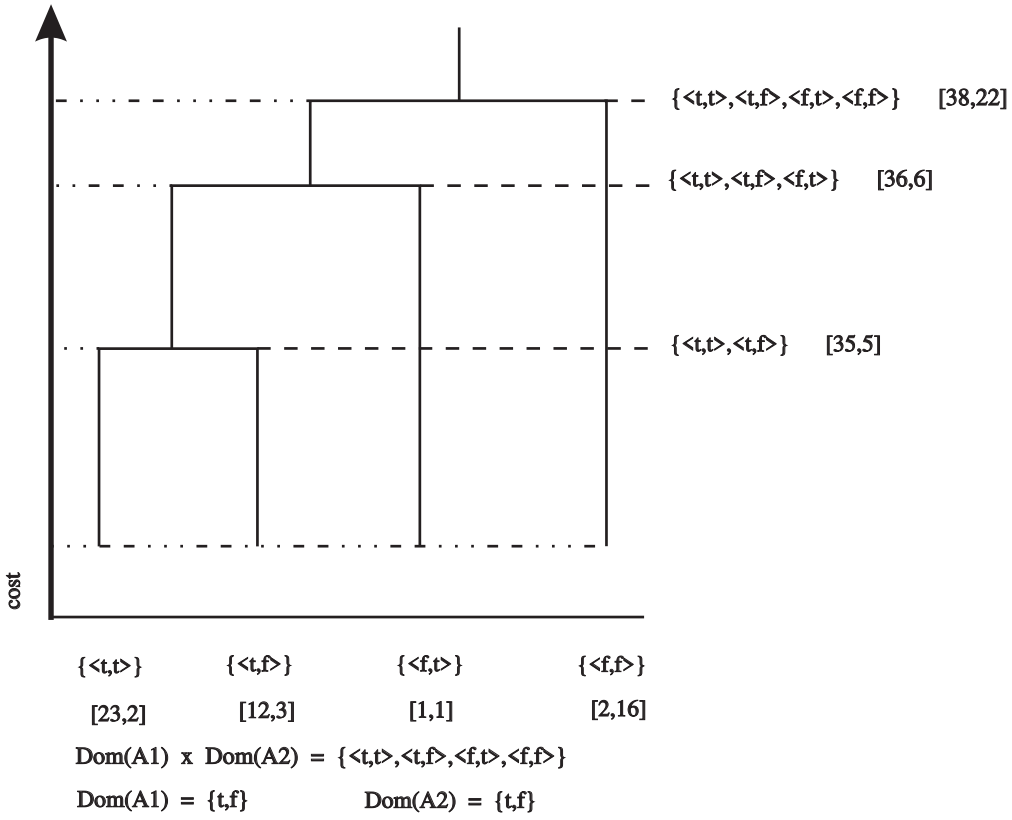

Figure 3: Hierarchical agglomerative cluster algorithm

The smallest increase of E is attained if clusters $\{< t, t >\}$ and $\{< t, f >\}$ are merged. The resulting partitioning is $c_2 = \{\{< t, t >, < t, f >\}, \{< f, t >\}, \{< f, f >\}\}$ and is evaluated as $E(c_2) = 6.67$. This procedure is repeated

until there are exactly two clusters left. The resulting partitioning corresponds to an extensional definition of a binary attribute. Let $c_3 = \{\{< t,t >, < t,f > , < f,t >\}, \{< f,f >\}\}$ be the final partitioning. The new attribute NA_1 is then defined as:

$$\begin{aligned}(NA_1 = t)\ iff\ & ((A_1 = t) \wedge (A_2 = t)) \vee \\ & ((A_1 = t) \wedge (A_2 = f)) \vee \\ & ((A_1 = f) \wedge (A_2 = t)) \\ (NA_1 = f)\ iff\ & ((A_1 = f) \wedge (A_2 = f))\end{aligned}$$

Obviously, the obtained extensional definition can "intensionally" be described as $((A_1 = t) \vee (A_2 = t))$.

After the application of the constructive operator o_κ to m operands, a single, best constructor according to E is turned into a new feature.

2.4.2 A*-Cluster

One possibility to achieve such a selection is a sequential performance of m cluster algorithms. However, CN2-MCI empoys a specific search strategy in order to save a number of cluster steps (i.e., merging of two clusters). A* [Hart et al., 1968] with a one-ply look-ahead is combined with beam search to construct m new features in parallel. To fit a hierarchical-agglomerative cluster algorithm into a search framework, each partitioning into clusters can be viewed as a node in an acyclic directed graph and each cluster step is treated as an expansion of a node. For each of the m cluster algorithms performed in parallel, a single, best node is kept in the set of open nodes. The best node for a single construction is determined by a bounded look-ahead of one ply. A*-Cluster proceeds as follows: The best node of all current constructions is selected. If it is a goal node (i.e., if it contains exactly two clusters), A*-Cluster stops with the optimal result of all constructions, because all the other partitionings of other constructions can only get worse. If it is not a goal node, the node is expanded and moved from the set of open nodes into the set of closed nodes. There exists exactly one successor of the selected node, because a one-ply look-ahead determines the best partitioning according to evaluation function E. The best possible subsequent partitioning according to E is inserted in the set of open nodes. The algorithm proceeds in this way until a goal node is found.
An A* evaluation function f is the sum of two functions $f(n) = g(n) + h(n)$. $g(n)$ evaluates the costs from the initial node to the actual node n and $h(n)$ is a lower bound for the costs to the cheapest goal node that can be reached from n. On the one hand, A*-cluster performs pure A*, but A* only "sees" one successor of a node and h is determined by a one-ply look-ahead. Therefore, problems of A* regarding memory complexity can be avoided. In fact, another perspective on A*-Cluster exhibits a beam-search with beam size m, where for each of the m

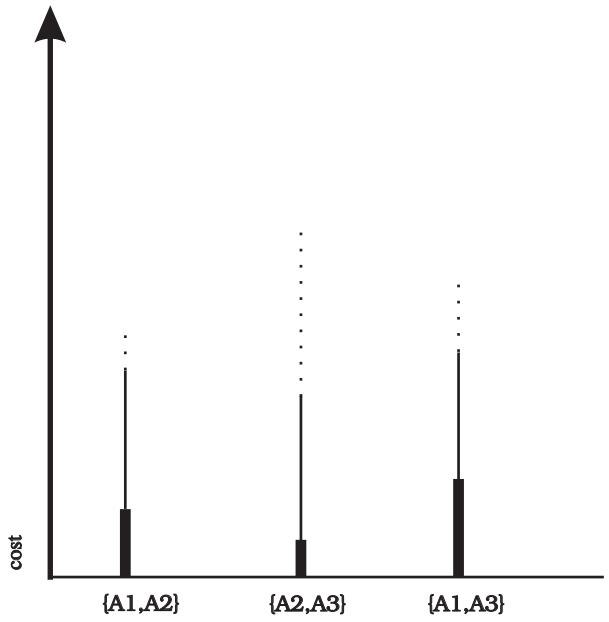

Figure 4: A*-Cluster

cluster processes the most promising node is kept in the beam. This is the reason why A*-Cluster cannot guarantee globally optimal solutions. The single cluster algorithms are still hill-climbing, while A*-Cluster determines the best result of these hill-climbing algorithms.

Example:

In Fig. 4, m is set to three. Thus, three pairs of attributes are selected as operands and three cluster algorithms are performed in parallel. Fig. 4 shows the values of the A* evaluation function $f(n) = g(n) + h(n)$. The "costs" of the actual partitionings (function g) are consisting of initial costs (depicted by bold lines) and the costs by subsequent cluster steps leading to the actual states (depicted by thin lines). The values of function h are depicted by dotted lines. The next node to be expanded is the one for attributes $\{A_1, A_2\}$, since the next cluster step obtains the best evaluation of all (according to f). If this step expands a goal node, A*-Cluster stops with an optimal result relative to the single hill-climbing cluster algorithms.

The overall approach is useful for several reasons:

- There are several paths to a globally optimal partitioning into two clusters. This is the reason why some minor sub-optimal decisions do not necessarily lead to a sub-optimal result. Thus, the hillclimbing approach does not deteriorate the quality of the partitionings too much.
- The h-part of an A*-function can be determined by means of a bounded look-ahead.
- A constant memory complexity is achieved through the beam-search aspect of A*-Cluster.
- The parallel construction provides a way to save cluster steps, although calculation of h by a one-ply look-ahead is essentially the search for an optimal successor.
- m operands are chosen in the first step of the selection, because an optimal selection of the operands does not necessarily lead to an optimal partitioning into two clusters. The result of the cluster algorithm is not known a priori.
- Evaluation is not problematic because of the "conservative" approach to feature construction. In every iteration of the algorithm, only the best feature that can be found is introduced. If the introduction of a new feature negatively effects the quality of the hypothesis, CN2-MCI puts the blame on the additional feature.

3 Empirical Results

3.1 Description of problem domains

3.1.1 Problem Monk1

The following attributes define the description spaces for the "monk's problems" [Thrun et al., 1991].

$$
\begin{aligned}
Dom(head\_shape) &= \{round,\ square,\ octagon\} \\
Dom(body\_shape) &= \{round,\ square,\ octagon\} \\
Dom(has\_tie) &= \{true,\ false\} \\
Dom(holding) &= \{sword,\ flag,\ balloon\} \\
Dom(jacket\_color) &= \{red,\ yellow,\ green,\ blue\} \\
Dom(is\_smiling) &= \{true,\ false\}
\end{aligned}
$$

The target concept for the monk1 problem can be described as follows:

(head_shape = body_shape) or (jacket_color = red)

The problem with the given representation is the non-existence of a CN2-operator for testing equality between features or an equivalent feature representing equality. A reasonable transformation should be achieved by means of a new feature representing equality.

3.1.2 Problem Monk2

The target concept of monk2 is defined as follows:

"exactly two of the six attributes take their first value"

Without some specific relations this target concept can only be given extensionally. The task of CN2-MCI will be to construct features which provide "access" to relations between variables for the selective induction algorithm. For instance, given features like *"head_shape and body_shape do not take their first value"*, the induced concept could be described more concisely.

3.1.3 Problem Monk3

The third problem domain is characterized by two major problems: Firstly, there is a classification noise level of 5 %. Secondly, CN2 does not include an operator for negating a selector.

The definition of Monk3 is given by:

(jacket_color = green and holding = sword) or

(not jacket_color = blue and not body_shape = octagon)

The problem with the original representation is in fact the problem of the selective learning module, which is incapable of expressing a negation. Apart from that, it is not clear how constructive induction operators can achieve an improvement in the presence of noisy data. For an explicit discussion of constructive induction in the presence of noise, see [Pfahringer, 1994b].

3.1.4 Problem Mux11

For each positive integer k there exists a multiplexer function which maps a tuple of $k + 2^k$ binary attributes into a binary attribute.

The first k attributes are address bits and the last attributes are data bits. The function takes the value of the data bit which is selected by the address bits. In the case of Mux11, $k = 3$, which yields an instance space defined by $3 + 2^3$ binary attributes.

Usual propositional learners cannot express this target concept concisely, because relations among attributes are essential for its description. Due to some peculiar properties of this problem, CN2-MCI is only partially able to remedy the problem with the originally given representation, although it constructs relations between data and address bits, as will be seen.

3.1.5 Problem Par5

For each positive integer k there exists an "even parity"-function, which maps a tuple of k binary attributes into a binary attribute. The function takes the value "true" if and only if the number of attributes taking "true" is even, else it takes "false".

The representational shortcomings are similar to the ones concerning the multiplexer-function. There is not only a need for features encoding relations, but generally also for new features computing some function out of the initial features. For instance, a new feature counting the numbers of those original features which take "true" would be useful for a simple concept description.

3.1.6 Overview of problem domains

Table 1 and Table 2 give a brief overview of the problem domains used for testing CN2-MCI.

	# training examples	# testing examples	# attributes	# classes
Monk1	124	308	6	2
Monk2	169	263	6	2
Monk3	122	310	6	2
Mux11	225	1823	11	2
Par5	25	7	5	2

Table 1: Overview of problem domains

	$\mu\|Dom(A_i)\|$	$\sigma\|Dom(Ai)\|$	class. noise level	# irrelevant attributes
Monk1	2.83	0.75	0	3
Monk2	2.83	0.75	0	0
Monk3	2.83	0.75	5	3
Mux11	2	0	0	0
Par5	2	0	0	0

Table 2: Overview of problem domains (continued)

3.2 Results of CN2-MCI in 5 problem domains

All results of the experiments were averaged over ten independent test runs. The example sets were randomly generated using the known target concept definitions. In such a way, more significant results can be achieved than with the original set of "monk"-examples used in [Thrun et al., 1991]. On the other hand, the results cannot be directly compared with the ones from the report.

First of all, a good setting of parameters was determined by tests in the Monk2 problem domain. Optimal results were obtained using 85 % of the training examples as primary training set and 15 % as secondary training set. Secondly, a rather large value of parameter τ turned out to be useful. In subsequent experiments, τ was set to 6.

In Table 3, the average accuracy of CN2-hypotheses is compared with the average accuracy of CN2-MCI-hypotheses. Both methods were applied to the same example sets. In all problem domains, CN2-MCI achieves better results than CN2 (see Table 3, last column). The best results are obtained with the first two monk-problems. In the case of Monk1, this is due to the construction of a binary feature representing equality in the first iteration of CN2-MCI:

	CN2-acc.	# new attrs.	CN2-MCI acc.	$\delta acc.$
Monk1	96.2	2.0	99.6	3.4
Monk2	90.6	10.7	95.3	4.7
Monk3	96.2	5.7	97.4	1.2
Mux11	82.9	1.8	85.7	2.8
Par5	68.3	4.6	71.5	3.2

Table 3: Results of CN2-MCI in 5 problem domains

$$new\_att_0 = true \Leftrightarrow head\_shape = body\_shape$$

Regarding Monk2, the number of iterations is significantly larger than for the other problems. This indicates a steady improvement of the hypothesis through the constructed features. When applied to the Monk2-problem, CN2-MCI in almost all cases constructs features like new_att_i:

$$new\_att_i = true \Leftrightarrow not\ att_j = first\_value\ and\ not\ att_k = first\_value$$

In most of the experiments in the Monk3 problem domain CN2-MCI defines the second disjunct as the first new feature. Further evaluation of the results indicates that constructive induction through CN2-MCI is not able to improve hypotheses in the presence of noise.

Interestingly, the method constructs few but effective features for the Mux11-problem. However, further transformations do not improve the representation. The structure of the multiplexer problem provides a variety of possible meaningful relations among data bits and selection bits. Therefore, the results in this domain are rather unexpected and require further analysis and explanation.

Results for Par5 indicate a mediocre performance of CN2-MCI in this domain. Nevertheless, most of the constructed features are intelligible. Most of the new features can be paraphrased by:

"The sum of bit_i and bit_j is 0"
"The sum of bit_i and bit_j is 1"
"The sum of bit_i and bit_j is 2"

4 Related Work

In this section a brief review of two closely related existing HCI-methods, FRINGE [Pagallo, 1989; Pagallo & Haussler, 1990] and AQ17-HCI [Wnek & Michalski, 1992a+b], will be given.

4.1 FRINGE

FRINGE [Pagallo, 1989; Pagallo & Haussler, 1990] constructs new attributes using structural information of the decision tree. Like CN2-MCI, this HCI-system introduces binary attributes in iterations. FRINGE searches for a repeatedly occurring pattern at the "fringe" of the decision tree, which indicates a representational shortcoming. In order to improve representation, FRINGE employs the pattern found (i.e. a subtree) as the definition of a new feature, where the values in the leaves are the values of the new feature. In subsequent iterations, the induced decision tree is significantly simplified by means of the new attributes.

4.2 AQ17-HCI and DUCE

Due to the relative lack of structure in decision rules, AQ17-HCI [Wnek & Michalski, 1992a+b] uses the most relevant disjuncts of a hypothesis as the definition of a new attribute. Like FRINGE, AQ17-HCI employs parts of the hypotheses as definitions of the constructed attributes. This kind of analysis of hypotheses is not capable of finding intelligible relations among variables, because the granularity of analysis is too coarse. To achieve a fine-grained analysis, a search for patterns occurring in the decision rules is necessary. Actually, DUCE [Muggleton, 1987] performs a search for such patterns, but again treats variables and their values as units, i.e. as binary attributes. Attributes taking different values in different rules are processed like different binary attributes. In other words, DUCE presupposes binary attributes as input, and therefore ignores the occurrence of the same attribute with a different value in another rule, which is potentially relevant to the construction process.

5 Discussion

In this section, we want to analyze and motivate some aspects of the method in more detail. Initially, the system was designed to achieve a more fine-grained analysis of decision rules. The expected benefits of such an analysis were already stated in the introduction. Many problems not concerning prior work like AQ17-HCI and DUCE arise, when the problem of a more fine-grained analysis of decision rules is tackled. Evidently, a more fine-grained analysis should involve a search for patterns occurring in the decision rules. The search for "optimal" patterns seems to be a hard task for several reasons. First of all it is not at all obvious which existing features the patterns should be composed of. Clearly, it is reasonable to presuppose a co-occurence of at least two features in more than one rule to designate it as a pattern. To simplify the task, a pattern is assumed to be composed of only two existing features. If a pattern consists of more than two features, this kind of regularity should be detected in a subsequent iteration.

Moreover, the values of the features involved should exhibit some regularity regarding the class attribute of the rule, i.e. the same values should occur in rules of the same class. The remaining task is to find a partitioning of the values of the 2-feature-pattern occurring in the hypothesis into as many disjoint subsets as there are values of the new feature. Furthermore and most importantly, the partitioning should be optimal or near-optimal according to a function which evaluates the regularity of the values of a pattern with respect to the rules in which they occur.

Thus, the problem of fine-grained analysis of decision rules is decomposed into two sub-tasks. The first one is to select the constructive operands by some heuristic which prefers pairs of attributes that are occurring in as many important rules as possible. A heuristic selection of operands appears plausible for one reason: If a pair of features does not occur in many important rules, it is unlikely to achieve a good result in the second step. Therefore it is reasonable to search for an occurrence of a pattern in many important rules. This procedure can be viewed as an abstraction from the actual attribute values, which nevertheless provides a necessary condition for finding relevant 2-feature-patterns in the generated inductive hypotheses. In the second step, the omitted details (attribute values and class membership of rules) of step one have to be considered again. The task is to find a "good" partitioning of the values of the 2-feature-pattern according to the evaluation function estimating the regularity with respect to the class attribute.

Initially, CN2-MCI was designed to search for patterns in decision rules like DUCE, but also processing attribute values. In later versions, the system was extended to process information from the training examples in the second step of the construction process. The access to the input data gives the algorithm a flavour of bottom-up (immediately generalizing input data) support of a (specializing) top-down rule induction algorithm. In other words, the partially data-driven construction of attributes enables the selective induction algorithm to perform a look-ahead of at least two original features. In this respect CN2-MCI differs significantly from HCI-systems.

6 Summary and Future Work

CN2-MCI is a purely empirical method and therefore well-suited if there is not much knowledge about the problem domain available. The method is designed to search for relations among variables, which are otherwise hidden to the learner. CN2-MCI can be applied either only to improve learning results, or to obtain a set of new descriptors. Nevertheless it should be noted, that these descriptors are constructed relative to the learning capabilities of a propositional selective induction algorithm. CN2-MCI can be viewed as generalization of FRINGE-like feature construction, where some major adaptations have to be made in order

to compensate the relative lack of structure in decision rules. Briefly, CN2-MCI can be described by two essential features: Firstly, the selection of m operands (i.e., pairs of features, from which a new feature is to be constructed) is based on co-occurrence of features in relevant rules. Secondly, CN2-MCI searches for a "good" partitioning of values of the operands into two clusters. The best resulting partitioning is selected as extensional definition of a new binary feature.

Tests with the Par5 and Mux11 problem domains suggest there is a possibility for an interesting extension of CN2-MCI: In these cases it might be a good idea to construct n-valued features, where n is determined in the course of the construction. Similar to ChiMerge [Kerber, 1992], CN2-MCI could decide how many values are an appropriate abstraction from the initial partitioning into clusters. In such a way, CN2-MCI could define n-valued features if some statitical tests suggest there is a real loss of information regarding the class attribute otherwise.

Furthermore, it would be interesting to involve background knowledge in the constructive induction process. Certainly, background knowledge would effect an improvement in intelligibility, which is still a problematic aspect of constructive induction.

Acknowledgments

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P8756-TEC. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Research. I would like to thank Gerhard Widmer for his encouragement, his patience, and his comments on earlier drafts of this paper. I also thank Hermann Kaindl for comments on an earlier version of this paper, and Herbert Dawid for helpful discussions on aspects of CN2-MCI.

7 Bibliography

[Bloedorn et al., 1993] E. Bloedorn, R.S. Michalski, J. Wnek: Multistrategy Constructive Induction: AQ17-MCI. *Proceedings of the* 2^{nd} *International Workshop on Multi-Strategy Learning*, Harper's Ferry, WV, 1993.

[Breiman et al., 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone: *Classification and Regression Trees.* Wadsworth & Brooks, Pacific Grove, CA, 1984.

[Clark & Boswell, 1991] P. Clark, R. Boswell: Rule induction with CN2: some recent improvements. *Proceedings of the* 5^{th} *European Working Session on Learning*, Porto. Springer Verlag, Berlin, 1991.

[Clark & Niblett, 1989] P. Clark, T. Niblett: The CN2 Induction Algorithm. *Machine Learning* 3:261-283, 1989.

[Eckes & Roßbach, 1980] T. Eckes, H. Roßbach: *Clusteranalysen.* Kohlhammer, Stuttgart, 1980.

[Hart et al., 1968] P.E. Hart, N.J. Nilsson, B. Raphael: A formal basis for the heuristic determination of minimum cost paths.*IEEE Transactions on SSC*, SSC-4:100-107, 1968.

[Jain, 1986] A.K. Jain: Cluster Analysis. in: *Handbook of Pattern Recognition and Image Processing.* T.Y. Young, K.S. Fu (Eds.), Academic Press, New York, 1986.

[Kerber, 1992] Randy Kerber: ChiMerge: Discretization of Numeric Attributes *Proceedings of the* 11^{th} *National Conference on Artificial Intelligence*, 1992.

[Langley et al., 1986] Pat Langley, Jan M. Zytkow, Herbert A. Simon, Gary L. Bradshaw: The Search for Regularity: Four Aspects of Scientific Discovery. in: *Machine Learning: An Artificial Intelligence Approach. Vol. II.* Morgan Kaufmann, San Mateo, CA, 1986.

[Lenat, 1976] D. B. Lenat: *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* Ph.D. dissertation. Stanford University, Stanford, CA, 1976.

[Matheus & Rendell, 1989] Christopher J. Matheus: Constructive Induction On Decision Trees.

Proceedings of the 11^{th} *International Conference on Artificial Intelligence*, 1989.

[Matheus, 1991] Christopher J. Matheus: The Need for Constructive Induction. *Proceedings of the* 8^{th} *International Conference on Machine Learning*, 1991.

[Michalski, 1983] Ryszard S. Michalski: A Theory and Methodology of Inductive Learning. in: *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Company, Palo Alto, CA, 1983.

[Michalski, 1993] Ryszard S. Michalski: Inferential Theory of Learning as a Conceptual Basis for Multi-strategy Learning. *Machine Learning*, 11, 111- 151, 1993.

[Muggleton, 1987] Steven Muggleton: DUCE, an oracle based approach to constructive induction. *Proceedings of the* 10^{th} *International Conference on Artificial Intelligence*, 1987.

[O'Rorke, 1982] P. O'Rorke: *A comparative study of inductive learning systems AQ11P and ID3 using a chess end-game test problem.* Technical Report ISG 82-2, University of Illinois, Computer Science Department, Urbana, 1982.

[Pagallo, 1989] Giulia Pagallo: Learning DNF by Decision Trees. *Proceedings of the* 11^{th} *International Conference on Artificial Intelligence*, 1989.

[Pagallo & Haussler, 1990] G. Pagallo, D. Haussler: Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5, pp. 71-97, 1990.

[Pfahringer, 1994a] Bernhard Pfahringer: Controlling Constructive Induction in CIPF: An MDL Approach. *Proceedings of the* 7^{th} *European Conference on Machine Learning*, 1994.

[Pfahringer, 1994b] Bernhard Pfahringer: Robust Constructive Induction *Report TR-94-11, Austrian Research Institute for Artificial Intelligence* Vienna, 1994.

[Rissanen, 1978] J. Rissanen: Modelling by Shortest Data Description. In: *Automatica, 14:465-471*, 1978.

[Thrun et al., 1991] S.B. Thrun et al.: *The MONK's Problems. A Performance Comparison of*

Different Learning Algorithms.
Technical Report, CMU-CS-91-197,
Carnegie Mellon University, 1991.

[Wnek & Michalski, 1992a] Janusz Wnek, Ryszard S. Michalski:
Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments.
MLI Report. Center for Artificial Intelligence,
George Mason University, 1992.

[Wnek & Michalski, 1992b] Janusz Wnek, Ryszard S. Michalski:
Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments.
MLI Report. Center for Artificial Intelligence,
George Mason University, 1992.

[Yang et.al. 1991] Der-Shung Yang, Larry Rendell, Gunnar Blix:
A Scheme for Feature Construction and
a Comparison of Empirical Methods.
Proceedings of the 12^{th} *International Conference on Artificial Intelligence*, 1991.