Robust Constructive Induction

Bernhard Pfahringer

Austrian Research Institute for Artificial Intelligence
Schottengasse 3
A-1010 Vienna
Austria

E-mail: bernhard@ai.univie.ac.at

Category: Paper

Area: Machine Learning

Keywords: Minimum Description Length, Constructive Induction, Noise.

Abstract

We describe how CIPF 2.0, a propositional constructive learner, can cope with both noise and representation mismatch in training examples simultaneously. CIPF 2.0 abilities stem from coupling the robust selective learner C4.5 (and its production rule generator) with a sophisticated constructive induction component. An important new general constructive operator incorporated into CIPF 2.0 is the *simplified Kramer operator* abstracting combinations of two attributes into a single new boolean attribute. The so-called *Minimum Description Length* (MDL) principle acts as a powerful control heuristic guiding search in the representation space through the abundance of opportunities for constructively adding new attributes. Claims are confirmed empirically by experiments in two artificial domains.

1 Introduction

When learning concept descriptions from preclassified examples, simple concept learners typically make strong assumptions about the way these examples are represented. For effectively learning a concept its examples must populate one or a few regions of the hypothesis space expressible in the description language. For example, decision trees encode axis-parallel nested hyper-rectangles. Two different problems may cause irregular distributions of learning examples in the original representation space: *noise* and/or an *inadequate description language*.

As a remedy for the latter problem constructive induction has been introduced, e.g. in [Dietterich & Michalski 81] and [Mehra et al. 89]. The basic idea is to somehow transform the original representation space into a space where the learning examples exhibit (more) regularities. Usually this is done by introducing new attributes and forgetting old ones. So constructive induction is searching for an adequate representation language for the learning task at hand.

Just like when learning from noisy examples, constructive induction must be controlled properly. Otherwise its application may yield too complex, convoluted induced concept descriptions which may be hard to understand and may perform poorly at predicting concept membership of unclassified examples. This phenomenon can be called *overfitting the representation language* in analogy to *fitting the noise*. The presence of both noise and an inadequate language obviously increases the possibilities for *overfitting* even further.

In [Pfahringer 94] we reported on our constructive learner CIPF and its relative success in noise-free domains due to rigid control applying the *Minimum Description Length* principle. This paper will concentrate on how the improvements found in the newest version CIPF 2.0 allow for *robust constructive induction* handling both noise and inadequate language. The main improvements over CIPF 1.0 are incorporating as the basic induction module a well-known sophisticated decision tree learner, namely C4.5 [Quinlan 93]; and including a new general operator for constructive induction, a simplified, more efficient version of the operator described in [Kramer 93].

Section 2 briefly describes a generic architecture for constructive induction and discusses CIPF 2.0 in these terms. In section 3 we will focus on how the problem of controlling search for useful changes of representation is solved in CIPF 2.0 by means of the powerful *Minimum Description Length (MDL) Principle* [Rissanen 78]. Section 4 describes the simplified *Kramer* operator. Experiments in two artificial domains - the *Monk's Problems* [Thrun et al. 91] and illegal king-rook-king chess positions [Fuernkranz 93] - are summarized in section 5. Section 6 draws conclusions, relates to various other approaches of constructive induction and talks about further research directions we are pursuing within CIPF.

2 A Generic Architecture and an Introduction to CIPF

This section will briefly describe a generic architecture for constructive induction and use this architecture to introduce CIPF. We will also discuss some important design rationales of CIPF.

Most implemented constructive induction systems can be described in terms of three different modules working together:

- A CI module: Given examples and attribute descriptions and possibly already some descriptions/hypotheses, this module constructs new attributes according to some methodology. Output of this module are new attribute descriptions and the augmented and transformed learning examples.
- A Selective Learner: Any (classical) propositional learning algorithm can be used to induce rules from the transformed learning data. Output of this module is a set of rules forming a hypothesis that compresses and explains the learning data.
- An Evaluator: This current hypothesis must be evaluated in some way to decide whether it is of good enough quality to serve as a final result, or if it should be input into another cycle of induction. It might also be the case that no good hypothesis is found, but computation nonetheless terminates due to exhausted resources like maximal number of cycles or heuristically/statistically based doubt about the possibility of finding any better hypothesis.

Actual systems not only differ in their choices for the different parameters (e.g. which methods they select for doing CI or what algorithm lies at the heart of their respective learner), they may even omit modules and/or pathways at all; for instance, some systems do not run in cycles, but perform sequential one-shot learning only.

The main goal in building CIPF is designing a practical system for constructive induction that minimizes the number of user-settable parameters. So we try to identify principled choices or automated ways of choosing good values for necessary decisions where other systems rely on user-specified parameter values. This was one reason for choosing the *Minimum Description Length Principle* as an evaluator. This will be described in more detail in the next section.

CIPF borrows heavily from existing systems in that we have tried to collect useful features of known machine learning systems. We try to combine these in a synergetic fashion in CIPF. CIPF is a true instance of the generic architecture for constructive induction described above in that it realizes all the boxes and pathways. CIPF's components will be detailed in the following.

2.1 Constructive Induction in CIPF (the CI Module)

Just like the multi-strategy system AQ17-MCI [Bloedorn et al. 93], CIPF takes an operator-based approach to constructive induction. It supplies a (still growing) list of generally useful CI operators plus an interface allowing for user-supplied special operators. For instance, these operators might encode possibly relevant background knowledge. We have currently implemented the following generally useful CI operators in CIPF 2.0:

- Compare attributes of the same type: is attribute `A1` *Equal to/Different from* attribute `A2`.
- Conjoin possible values of nominal attributes occuring in good rules into a subset of *useful* values for the respective attribute.
- Conjoin two attributes occuring in a good rule [Matheus & Rendell 89]. We use the simplified *Kramer* operator for this task as will be described in Section 4.
- For the set of positive examples covered by a good rule: compute subsets for the respective base-level attributes, so that these subsets exactly cover these positive examples.
- Drop attributes not used by any of the good rules. [1]

Recursive application of these operators may yield complex new attributes.

2.2 CIPF's Selective Learner

CIPF 1.0 used a simple propositional FOIL-like learner [Quinlan & Cameron-Jones 93], i.e. an original implementation of a simplified FOIL dealing with propositional horn clauses only. We preferred direct induction of rules over induction of decision trees mostly for two important reasons:

- Unknown values can be dealt with pragmatically: never incorporate tests for *unknown* in a rule.
- Induction focuses on one class at a time. At least in relational learning this approach seems to be superior to decision trees [Watanabe & Rendell 91] and we suspect that the same might be true for propositional learning.

[1] One might argue whether *dropping an attribute* really is a *constructive induction* operator or not. Anyway it being a very useful operator we have chosen to include it in the above list. Furthermore the terminology used in [Bloedorn et al. 93] defines the *set of constructive induction operators* as the union of *constructors* and *destructors*.

Due to pitfalls regarding CIPF 1.0's disabilities handling noise CIPF 2.0 now incorporates C4.5 [Quinlan 93] as its selective learner, a sophisticated decision tree algorithm well able to deal with noise. Fortunately C4.5 also includes a rule-generator transforming decision trees into sets of production rules. These rules are then processed and analyzed by CIPF 2.0's module for constructive induction. We use default settings for C4.5 in all experiments, so CIPF's selective learner essentially still is a parameter-less module, thus fulfilling one of our design criteria. Regarding the above-mentioned preference for production rules, reason a) is still true when using the output of C4.5RULES and reason b) is achieved by artificially turning an N-class learning task into N 2-class learning tasks.

3 Using MDL to Control Constructive Induction (the Evaluator Module)

CIPF takes a rather eager approach to constructive induction: at every step all possible new attributes are added. This over-abundance in the representation space may quickly results in unwieldy, overly complex induced rule sets or attributes when learning without appropriate control. Such rule sets ca be difficult to comprehend for the user and may yield mediocre results when classifying unseen examples. In analogy to *noise fitting* [Angluin & Laird 87] this phenomenon can be called *language fitting*. Typical examples of such behaviour are published in the section on AQ17-HCI in the *Monk report* [Thrun et al. 91], which describes three artificial learning problems for evaluating and comparing different algorithms. We have made similar experiences with early versions of CIPF lacking sophisticated control.

To prevent CIPF from *language* or *noise fitting* we have devised the following simple, yet effective control regime:

- Every time the CI module is called, it is allowed to construct a more or less unlimited number of new attributes.
- These attributes will be input to the next learning step. There they will *compete* with each other for being used in induced rules.
- Only the *fittest* attributes will be allowed to survive.

So how are the *fittest* attributes determined in CIPF? We pragmatically equate these with the set of attributes being used by *good* rules. Right now *good* rules is pragmatically defined as just the rule-set returned by C4.5RULES as C4.5's rule generator does a good job of generalizing/pruning both single rules and complete rule-sets constructed out a decision tree. So one basic step in CIPF consists of the following actions:

- Express the training examples in terms of the currently active attributes, not distinguishing between base-level or constructed attributes).
- Call C4.5 on this training set, and call C4.5RULES to generate production rules out of the decision tree.
- Analyze the set of production rules to a) determine surviving *old* attributes and b) constructing new attributes via some CI-operator.

How does overall control work, how many such basic steps are taken? CIPF at the moment just iterates as long as new attributes are introduced. It keeps track of dropped attributes and never introduces an attribute twice, so this criterion leads to termination in between a few and a few dozen cycles, depending on both the amount of noise and the mismatch of the representation language.

Now we can answer the question of which rule set will be chosen as the final result of induction. Instead of using some ad-hoc measures of accuracy and quality or some user-supplied evaluation functions we have identified the so-called *Minimum Description Length Principle* [Rissanen 78, Quinlan & Rivest 89] as a very well-performing evaluator when it comes to choosing the one *best* rule-set from all the induced rule-sets.

In a nutshell, MDL is a concept from information theory that takes into account both a theory's simplicity and a theory's predictive accuracy simultaneously. MDL is disarmingly simple: concept membership of each training example is to be communicated from a sender to a receiver. Both know all examples and all attributes used to describe the examples. Now what is being transmitted is a theory (set of rules) describing the concept and, if necessary, explicitly all positive examples not covered by the theory (the false-negative examples) and all negative examples erroneously covered by the theory (the false-positive examples). Now the cost of a transmission is equivalent to the number of bits needed to encode a theory plus its exceptions in a sensible scheme. The MDL Principle states that the best theory derivable from the training data will be the one requiring the minimum number of bits.

So for any set of rules generated by the selective learner a *cost* can be computed. The rule-set with minimum cost is supposed to be (and in the experiments reported below most often really is) the best theory for the training data. The precise formula used to apply the MDL Principle in CIPF is the same one as used by C4.5 [Quinlan 93] for simplifying rule sets:

$$Cost = TheoryCost + log_2\left(\left(\begin{array}{c} C \\ FP \end{array}\right)\right) + log_2\left(\left(\begin{array}{c} NC \\ FN \end{array}\right)\right)$$

In this formula `TheoryCost` is an estimate for the number of bits needed to encode the theory. `C` is the total number of training examples covered by the theory, `FP` is the number of false-positive examples, `NC` is the total number

of training examples not covered by the theory, and `FN` is the number of false-negative examples. So the second and the third term of the formula estimate the number of bits needed to encode all false-positive and all false-negative examples respectively. In summary this formula approximates the total cost in number of bits for transmitting a theory and its exceptions. As in C4.5 the actual implementation uses a weighted sum of both the theory and the exception cost. These weights have been hard-wired into CIPF after some initial experiments and are set to one and three for theory and exceptions respectively.

It is of course necessary to also assess and include cost for constructed attributes into the total cost of theories. Otherwise CIPF would successively create more and more complex attributes probably only stopping at a theory of exactly two rules testing a single boolean attribute: if true we conclude class A, if false we conclude class B. This way the complexity would solely be shifted from the rules into the structure of the attributes (and would be hidden there at zero cost).

To prevent this from happening, CIPF assesses cost of constructed attributes by estimating the cost of encoding such constructions (expressed in numbers of bits). For example a subset test (attribute A's value is one of the following values) needs to encode the kind of attribute (subset test), the underlying attribute being involved (A), the set of values (possibly as a bitmask for the total set of possible values for attribute A), and an additional bit to represent whether we test for truth or falsity Thus using a *constructed* attribute entails a kind of penalty or cost, which can be amortized either if this attribute offers superior compression or if it is used in more than one rule.

Empirically this simple strategy seems to produce good results, as indicated by the experiments reported in section 5 and it is effectively computable. Also, to repeat its two main advantages, the strategy includes no user-settable parameters, and it also does not require a secondary training set (*train-test set*) to evaluate the quality of constructed attributes, like e.g. AQ17-MCI or various forms of *reduced error pruning* [Fuernkranz & Widmer 94] do.

4 The Simplified Kramer Operator

In [Kramer 93] a new, general constructive induction operator is introduced, which essentially abstracts the extensional product of the set of possible values of two given attributes to a new boolean attribute. For example two attributes A1 with possible values a or b and A2 with possible values 1, 2, or 3 could be abstracted to C(A1,A2) as:

```
C(A1,A2) = t iff A1 = a and A2 is 1 or 2
C(A1,A2) = f iff A1 = b and A2 is 3
```

When applied to two boolean attributes, the result can of course be any binary boolean function (including e.g. xor or nand). We take care of immediately

rejecting trivial constructions like tautologies or projections of one of the arguments.

CIPF 2.0 introduces the following simplifications. In [Kramer 93] a heuristic chooses a few good rules and from these rules a few pairs of co-occuring attributes are taken as input for a involved A*-search for the best split according to another heuristic estimating split values. On the contrary, CIPF uses all pairs of co-occuring attributes, estimates for each such pair its info-gain on the original training set, and introduces the single (non-trivial) abstraction with the highest info-gain. Info-gain and binarization for abstraction are computed greedily as follows:

- For each pair of values of the two attributes determine the number of positive and negative examples covered by these two tests.
- Sort all pairs according to the ratio of positive versus all examples covered.
- Compute info-gain for all split-points in this sorted list of pairs.
- Finally choose the split yielding the maximal info-gain.

So this operator can also be partially described as compiling limited look-ahead information for decision tree induction into new attributes. Our simplifications allow for efficient implementation with tightly limited search (linear in the number of pairs of values), but still seems to yield useful (though maybe not always immediately the absolute best) abstractions. This operator can also be seen as a generalization of the technique described in [Breiman et al. 84] for computing optimal binary splits for *single* attributes: we handle combinations of two attributes. The usefulness of this operator is also evident in the results of the experiments reported on below.

5 Experiments

In the following experiments, CIPF 2.0's performance was usually averaged over ten runs randomly choosing the appropriate number of training examples and randomly reversing the class attribute (from *yes* to *no* or vice versa) for N% of the these examples, when the noise level was set to N. We always report testing accuracy of the initially induced rule-set (which is of course identical to the result of just calling C4.5 followed by C4.5RULES) and the accuracy of the *best* set according to the MDL-heuristic used. Typically this *best* accuracy as selected by the MDL principle is also the absolute best value of any of the induced rule-sets. Occasionally though, for combinations of higher noise levels with a low percentage of examples selected for training CIPF 2.0 failes to choose the best possible candidate rule-set. Nonetheless the final result is most often better than the initial result, and for the rare cases where it is not, differences are marginal, e.g. an accuracy of 80% instead of 82%.

5.1 Monk's Problems

The *Monk's problems* [Thrun et al. 91] are three artificially constructed problems in a space formed by six nominal attributes having from two to four possible values. There is a total of 432 different possible examples. The three problems are abbreviated to Monk1, Monk2, and Monk3 in the following. CIPF 2.0 results for the Monk's problems very encouraging. The original Monk's Problems' definition explicitly specifies a training set for each of the three problems and measures accuracy on the total set of possible examples. From table 1 we see that CIPF 2.0 solves all three problems satisfactorily.

Monk1 was solved without problems. CIPF finds the correct theory:

```
true <= (jacket_color = red)

true <= (head_shape = body_shape)
```

This is no surprise as this example is simple and CIPF has the necessary constructive operator *compare attributes of the same type* at its disposal. Furthermore, regarding solely accuracy, even the initial rule-set is a 100% accurate; but when (implicitly) rewritten by constructive induction to the above given concept definition, it is much more concise and therefore it correctly gets assigned a lower MDL estimate.

Performance on Monk2 shows how effective constructive induction can be. Starting from an accuracy of only 67.1% CIPF 2.0 manages to induce an almost correct concept definition giving an accuracy of 95.6%. This success can be attributed to the addition of the simplified *Kramer* operator as described in section 4. This general constructive operator is (after repeated application in 11 cycles of induction and construction) able to compute a very good approximation of the correct theory.

Results for Monk3 are quite good, shoowing that CIPF 2.0 has overcome its initial problems regarding noise. This is of course the result of incorporating such a robust induction algorithm like C4.5 into CIPF. Actually C4.5 alone is able to solve this problem to full 100% accuracy when called with the `-s` flag to force subsetting of nominal attributes. But this flag is not set as a default because it can cause large computational overheads and it is speculated, that a more focussed way of introducing subsets of possible values might be more useful. At least for Monk3 CIPF 2.0 seems to prove this speculation, as its constructive operators allow for the introduction of the right subsets necessary to improve accuracy from the intial 96.3% to the final 100%.

Additionally we would like to mention that some other learning systems also exhibit very good performance on the original *Monk's problems*, e.g. AQ17-HCI achieves 100%, 93.1%, and 100% on Monk1, Monk2, and Monk3 respectively, [2]

[2] AQ17-HCI has at its disposal a very special CI operator which perfectly fits the Monk2 problem, thus explaining its impressive performance on this problem.

	CiPF first	CiPF best
Monk1	100.0	100.0
Monk2	67.1	95.6
Monk3	96.3	100.0

Table 1: Monk's Problems: accuracies (percentages) for CiPF after the first and after the best cycle of induction.

and a specialized form of Backpropagation yields 100%, 100%, and 97.2% respectively.

As there are fixed, prespecified training and test sets for the Monk's problems, there is of course always the danger lurking that one tunes one's system to good performance on these sets. To prevent us from this pitfall and to a larger degree to study *CiPF 2.0*'s abilities regarding noise and inadequate representations simultaneously, we designed the following series of experiments.

For Monk1 we randomly chose 30, 40, 50, or 60% of all examples for training, chose a noise level of 0, 5, 10, 15, 20, or 25% respectively and for every combination did ten test-runs of CiPF to average results. These averages are given in table 2.

Interpreting table 2 we can notice a few interesting facts regarding Monk1: certainly results get more flakey when both noise is high and the number of training examples is small. Still in the worst case of 25% noise and only 30% percent training examples CiPF 2.0 still on average achieves an accuracy of 81.5%. Furthermore, on average the final result is never worse than the initial accuracy, and if both values are equal, then these values are also high. C4.5 alone proves to be quite a robust learner given medium or smaller levels of noise and adequate numbers of examples: the initial runs for these cases always yield accuracies around $(100 - Noise/2)\%$ or better. Still in almost all cases constructive induction significantly improves the final result.

For Monk2 we deliberatly left columns for 30 and 40% training set size empty, because results looked more or less like for the 50% experiments: final accuracies are rarely significantly better than the average initial 65%. For constructive induction to reliably push overall accuracy to levels above 90% we need both an adequate number of training examples and a moderate noise level. This is due to the rather complicated target concept definition: it is a kind of an xor including *all* base-level attributes testing if exactly two of these six attributes have as their respective value the first possible value given in their domains. But given sufficient information CiPF 2.0 is able to achieve impressive improvements. Other approaches seem to achieve such improvements only by incorporating specialized CI operators fitting well this special kind of target concept.

TrainEx%	Noise%	Monk1		Monk2		Monk3	
		first	best	first	best	first	best
30	0	99.1	100.0			99.9	100.0
	5	97.4	100.0			98.7	99.1
	10	93.7	99.4			97.0	98.4
	15	87.4	96.6			93.8	93.4
	20	80.4	89.2			90.0	90.4
	25	77.8	81.5			84.3	85.7
40	0	99.4	100.0			100.0	100.0
	5	98.8	99.9			99.4	100.0
	10	97.8	99.4			97.0	98.3
	15	92.7	97.7			95.5	98.3
	20	89.5	97.9			89.0	95.4
	25	78.9	88.9			84.4	91.9
50	0	100.0	100.0	68.2	79.8	100.0	100.0
	5	99.7	100.0	66.5	87.1	99.3	99.8
	10	96.9	99.7	67.8	84.8	97.3	99.8
	15	92.9	99.0	63.7	64.1	95.1	98.0
	20	88.0	96.3	63.2	63.4	93.3	97.8
	25	83.8	95.8	60.2	60.9	87.7	94.1
60	0	100.0	100.0	64.5	97.3	100.0	100.0
	5	99.8	99.8	66.1	83.6	99.6	99.9
	10	96.7	99.6	63.7	81.7	97.6	99.2
	15	93.1	98.8	64.1	65.2	95.9	99.1
	20	91.1	97.9	63.6	63.6	90.5	97.5
	25	85.2	99.0	60.7	65.4	87.3	94.7

Table 2: Monk's Problems: accuracies (percentages) for CIPF 2.0 after the first and after the best cycle of induction for various levels of noise and various sizes of the training set for Monk1, Monk2, and Monk3.

For Monk3 we can more or less repeat the facts found for Monk1, with overall final accuracies being even better: with the exception of the 25% noise and 30% training examples case every final accuracy is above 90%.

This test series empirically proves the utility of constructive induction even when "only" dealing with noise. Constructive induction seems to shift the representation language towards appropriate, more concise definitions allowing the learner to distinguish more easily between variety and noise.

5.2 Illegal King-Rook-King Chess Positions

This domain is a very valuable testbed for experiments involving various amounts of noise and varying sizes of training sets. As such it has been used intensivly in inductive logic programming. There are a few hundred thousand different possible examples. [Fuernkranz 93] is a theoretical study including various approximate theories and showing a test-set size of 5000 to be sufficient for estimating accurracies of induced theories.

KRK is very easily represented for CIPF. The original example tupels of the relation `illegal/6` have six arguments encoding rank and file of all three pieces. Background knowledge in the original formulation consists of definitions for `=/2`, `less_than/2` and `adjacent/2`. Taking into regard predicate modes, symmetries, and the fact that in CIPF with a boolean attribute both a test and its negation can be represented, every original `illegal/6` example was transformed into a tuple of 18 boolean attributes encoding all possible body literals.

Induced theories (with no noise present) usually resemble the approximate theories given in [Fuernkranz 93]. A sample theory derived by CIPF from 100 training examples looks as follows:

```
[1] illegal <= (BLACK-KING-FILE = WHITE-ROOK-FILE)

[2] illegal <= (BLACK-KING-RANK = WHITE-ROOK-RANK)

[3] illegal <= (adjacent BLACK-KING-FILE WHITE-KING-FILE) and
               (adjacent BLACK-KING-RANK WHITE-KING-RANK)

[4] illegal <= (adjacent BLACK-KING-FILE WHITE-KING-FILE) and
               (BLACK-KING-RANK = WHITE-KING-RANK)
```

This approximate theory was tested with 5000 test examples yielding an accuracy of 98.4%. This is consistent with [Fuernkranz 93] which proves a theory consisting of the first three clauses 1,2,3 to be 98.451% correct.

To get a better picture of the relationship between class noise, training set size, and testing accuracy, we ran experiments using training sets of 100, 250, and 500 examples, choosing noise levels of 0, 10 and 20% each and randomly iterating five times for each pair of settings. Averaged accuracies are given in table 3.

Noise%	100 Ex		250 Ex		500 Ex	
	first	best	first	best	first	best
0	98.6	98.6	98.5	98.6	99.1	99.3
10	95.0	95.7	97.1	97.5	98.5	98.9
20	92.4	93.2	96.2	96.5	97.2	97.8

Table 3: Illegal KRK: accuracies (percentages) for CIPF 2.0 after the first and after the best cycle of induction for various levels of noise and various sizes of the training set in the King-Rook-King domain.

Using only 100 training examples at a noise level of 10%, CIPF 2.0 significantly outperforms all approaches compared in [Fuernkranz & Widmer 94]. When using 250 training examples, it performs slightly worse than the best approach (incremental reduced error pruning - IREP) cited in [Fuernkranz & Widmer 94]. For 500 training example CIPF 2.0 in turn outperforms IREP by an even smaller margin: 98.48% vs. 98.9%. These small differences for training set sizes of 250 or 500 examples may not be statistically significant, though.

On the overall the expected effect - larger absolute error - can be found when dealing with small example sets at higher noise levels. The absolute differences are rather small, though. Generally, for this domain the selective learner on its own produces almost perfect theories. Therefore constructive induction is only occasionally able to improve the scores marginally. But in every test-run the initially induced theory was rewritten into a concise and easily comprehensible form like exemplified by the above given sample rule-set.

6 Conclusions, Related Work, and Further Research

We have shown empirically, that interfacing a robust selective learner to strong constructive operators under a rigid control schema can result in a robust constructive induction system being able to deal with both noise and inadequate representation language simultanously. Incorporating the MDL Principle into CIPF as the single, uniform heuristic for evaluating theories and thereby implicitly guiding constructive induction proved valuable. The MDL Principle combines both accuracy and complexity of a theory into a single uniform measure. Thus CIPF does not require any ad-hoc measurements or user-defined evaluation functions of possibly questionable quality and can nonetheless use *all* of the available training data for induction. Other approaches (e.g. AQ17-MCI or reduced error

pruning) have to resort to splitting the training data into two or more sub-parts performing some sort of cross-validation on these sub-parts. Such an approach may be more expensive computationally (but see [Fuernkranz & Widmer 94] for efficient reduced error pruning) and may miss regularities in the data for reasons intrinsic to this approach. Still, on a systems level, CIPF certainly is most closely related to and influenced by the multi-strategy system AQ17-MCI. The main differences are the underlying inductive learner and the way control is imposed on constructive induction. CIPF eagerly tries to use every opportunity for constructive induction until the MDL principle helps choosing the best result from this cycling process. AQ17-MCI takes a different approach: relying on a set of meta-rules [Aha 92], it tries to identify the need (*when*) and the directions (*how*) for a change in the representation space. On the operator side AQ17-MCI seems to be more mature especially regarding so-called *deconstructors*. It would certainly be interesting to compare both systems on some tasks using the same set of operators in both systems.

Principled Constructive Induction is an interesting concept introduced in [Mehra et al. 89]. Geometric interpretation of the various constructors and the notion of *linear separability* is used to guide the selection of appropriate constructors. These ideas might have interesting implications for CIPF, too.

The problem of *language fitting* is also mentioned and discussed in [Matheus 90] in the context of the CITRE system and a framework for constructive induction. This approach uses additional background knowledge in two different ways when constructing attributes. Domain-knowledge constraints are used to eliminate less desirable new attributes beforehand and domain-dependent transformations generalize newly constructed attributes even further in ways meaningful to the current problem. Though these ideas do not currently fit directly into CIPF's schema for constructive induction, they might still point to valuable further improvements possible for CIPF.

Our further research directions for CIPF 2.0 include adding again support for numerical attributes (as was already present in CIPF 1.0, application to especially medical databases, and the definition of constructive operators dealing with structured objects, hopefully giving *CiPF* some of the representational power found in inductive logic programming systems like FOIL.

Acknowledgements

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 constructive discussion and help with this paper, and Johannes Fürnkranz for providing the king-rook-king position generator.

References

[Aha 92] Aha D.W.: Generalizing from Case Studies: A Case Study, in Sleeman D. and Edwards P.(eds.), Machine Learning: Proceedings of the Ninth International Workshop (ML92), Morgan Kaufmann, San Mateo, CA, pp.1-10, 1992.

[Angluin & Laird 87] Angluin D., Laird P.: Learning from Noisy Examples, Machine Learning, 2(4), 343-370, 1987.

[Bloedorn et al. 93] Bloedorn E., Wnek J., Michalski R.S.: Multistrategy Constructive Induction: AQ17-MCI, in Michalski R.S. and Tecuci G.(eds.), Proceedings of the Second International Workshop on Multistrategy Learning (MSL-93), Harpers Ferry, W.VA., pp.188-206, 1993.

[Breiman et al. 84] Breiman L., Friedman J.H., Olshen R.A., Stone C.J.: Classification and Regression Trees, Wadsworth International Group, Belmont, CA, The Wadsworth Statistics/Probability Series, 1984.

[Dietterich & Michalski 81] Dietterich T.G., Michalski R.S.: Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods, Artificial Intelligence, 16(3), 257-294, 1981.

[Fuernkranz 93] Fuernkranz J.: A numerical analysis of the KRK domain. Working Note, 1993. Available upon request.

[Fuernkranz & Widmer 94] Fuernkranz J., Widmer G.: Incremental Reduced Error Pruning. OeFAI Tech Report TR-94-09, also submitted to ML-94, 1994.

[Kramer 93] Kramer S.: CN2-MCI: Ein zweistufiges Verfahren für konstruktive Induktion, Master's thesis in preparation, Vienna, 1993.

[Matheus & Rendell 89] Matheus C.J., Rendell L.A.: Constructive Induction On Decision Trees, in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Morgan Kaufmann, Los Altos, CA, 645-650, 1989.

[Matheus 90] Matheus C.J.: Adding Domain Knowledge to SBL Through Feature Construction, in Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI -90), AAAI Press/MIT Press, Menlo Park, CA, pp.803-808, 1990.

[Mehra et al. 89] Mehra P., Rendell L.A., Wah B.W.: Principled Constructive Induction, in Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89), Morgan Kaufmann, Los Altos, CA, 651-656, 1989.

[Pfahringer 94] Pfahringer B.: Controlling Constructive Induction in CiPF: An MDL Approach, in Proceedings of the European Conference on Machine Learning (ECML94), 1994.

[Quinlan & Rivest 89] Quinlan J.R, Rivest R.L.: Inferring Decision Trees using the Minimum Description Length Principle, in Information and Computation, 80:227-248, 1989.

[Quinlan & Cameron-Jones 93] Quinlan J.R., Cameron-Jones R.M.: FOIL: A Midterm Report, in Brazdil P.B.(ed.), Machine Learning: ECML-93, Springer, Berlin, pp.3-20, 1993.

[Quinlan 93] Quinlan J.R.: C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA, 1993.

[Rissanen 78] Rissanen J.: Modeling by Shortest Data Description, in Automatica, 14:465-471, 1978.

[Thrun et al. 91] Thrun S.B., et.al.: The MONK's Problems: A Performance Comparison of Different Learning Algorithms, CMU Tech Report, CMU-CS-91-197, 1991.

[Watanabe & Rendell 91] Watanabe L., Rendell L.: Learning Structural Decision Trees from Examples, in Proceedings of the 12th International Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp.770-776, 1991.