A Multistrategy Learning System and Its Integration into an Interactive Floorplanning Tool

Jürgen Herrmann, Reiner Ackermann, Jörg Peters, Detlef Reipa

Universität Dortmund, Informatik I
44221 Dortmund, Germany
herrmann@ls1.informatik.uni-dortmund.de

Keywords: learning and problem solving, applications of machine learning, multistrategy learning

Abstract. The presented system COSIMA learns floorplanning rules from structural descriptions incrementally, using a number of cooperating machine learning strategies: Selective inductive generalization generates most specific generalizations using predicate weights to select the best one heuristically. The predicate weights are adjusted statistically. Inductive specialization eliminates overgeneralizations. Constructive induction improves the learning process in several ways. The system is organized as a learning apprentice system. It provides an interactive design tool and can automate single floorplanning steps.

1. Introduction

During the last few years the number of reported machine learning applications to real-world problems has grown significantly. ML techniques have been used successfully for tasks from various domains (Morik, 1992), (Kodratoff and Langley, 1993). A main use for the implemented systems is the acquisition of knowledge and data for knowledge-based systems and for conventional software systems.
Many successful machine learning systems acquiring knowledge about complex real-world problems are shaped according to characteristics of the considered domain, i.e. they are tailored to an application. In this context there are two relevant directions of machine learning research:

- The combination of several different machine learning strategies that cooperatively acquire knowledge about the domain (multistrategy learning systems, (Michalski, 1993; Saitta et al, 1993; Morik, 1993))
- The integration of a learning system into a problem solving tool for the domain (van Someren, 1993)

In this paper we present the learning tool COSIMA that combines these two aspects. The system acquires knowledge about floorplanning, a subtask of IC design. The incrementally learned rules are used to automate single design steps performed with an interactive floorplanning tool. For learning the following strategies are used: Selective inductive generalization, inductive specialization, constructive induction and statistical adjustment of parameter weights.
The rest of the paper is organized in following way: To motivate the design decisions for COSIMA, we briefly introduce the floorplanning domain in Section 2. Section 3 gives on overview on the COSIMA system. The combination of the different learning strategies is explained. Our multi-staged inductive generalization algorithm is presented in Section 4. Section 5 deals with the inductive specialization component that is used to eliminate

overgeneralizations. The three different uses of constructive induction in COSIMA are described in Section 6. The mechanism for the tuning of predicate weights (which influence all other learning strategies) is the topic of Section 7. Some remarks about the representation of floorplanning knowledge can be found in Section 8. COSIMA's problem solving component that is organized as a floorplanning assistant is explained in Section 9. A description of results and conclusions are presented in the last two sections.

2. Characteristics of the Floorplanning Domain

The system COSIMA acquires knowledge about floorplanning for the early phases of IC design. Floorplanning synthesizes a geometrical hardware description from a structural one (a netlist of functional blocks and their interconnections on register-transfer level). The functional blocks are placed on a two-dimensional area and connected to each other. Typically, the placement is performed by the designer stepwise (see Figure 1).

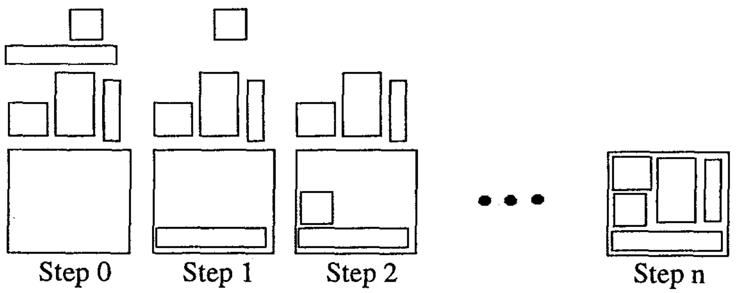

Figure 1: Simplified sequence of steps performed during floorplanning

The objectives of this task are the minimization of the consumed chip-area and the minimization of the total length for the connections. A machine learning system for the floorplanning domain has to be shaped according to the following characteristics:

- There is only *limited and incomplete knowledge* about synthesis and analysis of floorplans. As a consequence machine learning strategies that depend on a strong domain theory cannot be applied. Besides that, the evaluation of a performed floorplanning step is difficult: The effect of an operation on the quality of a floorplan can be evaluated accurately only at the end of a floorplanning process. So a "good-looking" operation can turn out to have a negative effect later. Because of the incomplete knowledge about floorplan analysis, overgeneralizations cannot be avoided.
- The learning and representation of *topological and geometrical properties* is crucial. To represent this information adequately, structural descriptions with typically several hundred facts are required. Therefore, a learning system for floorplanning must scale up - it must be capable to with cope with complex example descriptions. For the same reason, it is not feasible to store old examples - instead incremental learning should be used.
- Depending on the state of the floorplanning process *different aspects of an example have diverging importance.* To represent this fact, we use weighted predicates (see Section 4).

3. Combination of Different Machine Learning Strategies

The floorplanning characteristics described above have led to the selection and combination of the different machine learning strategies for COSIMA described in this section.
Selective inductive generalization is the basic learning strategy used to generate floorplanning rules from examples. An example consists of a floorplanning state (the floorplanning basic area on which some blocks are placed and the list of unplaced blocks) and an operator to be applied to a certain part of this state description. COSIMA uses a fixed set of predefined basic floorplanning operators. A floorplanning rule consists of a right-hand side determining the operator to be applied and a left-hand describing the situations in which the operator can be applied successfully (according to the objectives). Each time a new example becomes available, the corresponding rule is generalized.
Overgeneralizations are eliminated by use of *selective inductive specialization.* An overgenerlization is detected, if a rule matches on a negative example. If an example is rejected by COSIMA's built-in evaluation function or if it is classified by the user accordingly, it is marked as a negative example. The evaluation function can reject an example, if the applied operator leads to an obvious decrease of the floorplan quality.

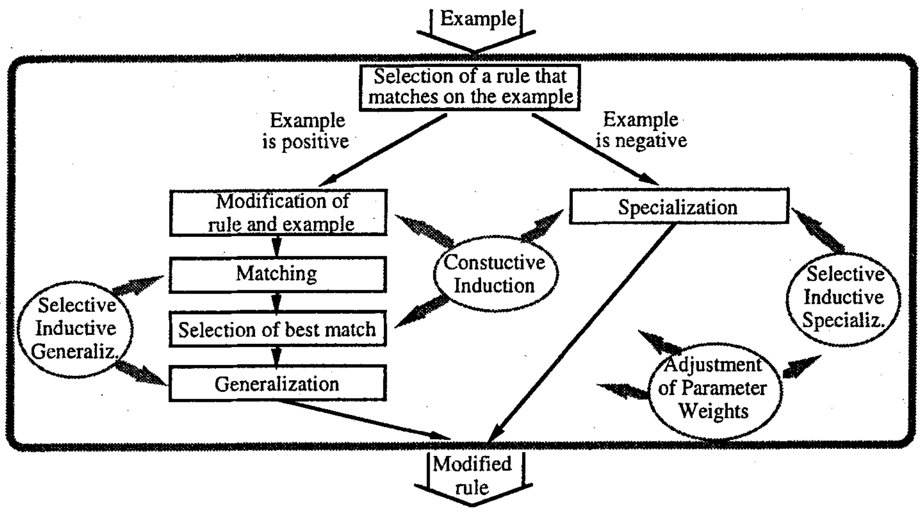
Example
Selection of a rule that matches on the example
Example is positive
Example is negative
Modification of rule and example
Matching
Selection of best match
Generalization
Selective Inductive Generaliz.
Constuctive Induction
Specialization
Selective Inductive Specializ.
Adjustment of Parameter Weights
Modified rule

Figure 2: Cooperation of different machine learning strategies in the COSIMA system. Selective inductive generalization and constructive induction cooperatively incorporate a positive example into an existing rule. Inductive specialization and constructive induction are used to eliminate overgeneralizations. Parameter adjustment influences all other strategies.

Constructive induction is used for three different purposes:

1) Before each generalization step, the descriptions of the new example and the corresponding rule are modified to improve generalization.
2) Typically, each generalization step results in several alternative most specific generalizations (MSGs). New predicates are constructed and added to the MSGs to select the best one heuristically.
3) If the selective inductive specialization fails to eliminate an overgenerization, constructive induction is used as an alternative specialization strategy. This situation occurs, if a correct discrimination of a negative example is not possible with current hypothesis language.

Predefined predicate weights guide COSIMA's three different inductive learning strategies. On the basis of statistics about sequences of generalization steps *adjustment of predicate weights* is performed.
The cooperation of the strategies is depicted in Figure 2. In the next sections the different strategies are described with more detail.

4. Selective Inductive Generalization

As COSIMA is learning from examples *incrementally*, each generalization step matches *two* example descriptions (or one rule and one example). Typically each description consists of several hundred facts. As Haussler (Haussler, 1989) has pointed out many MSGs (of different quality) may be created from structural descriptions for complex design tasks. A mechanism is therefore necessary to evaluate (intermediate and final) generalizations and to select the best one. The quality of a generalization depends on the corresponding facts in both input descriptions in accordance with a certain list of consistent object bindings[1]. As far as complex design tasks are concerned, each description typically consists of some important facts and many additional ones describing detail information. A good MSG must prefer the important ones, i.e. the selection of the object bindings must be dominated by the important facts.

Example: Two different lists of object bindings

Many possible combinations of object bindings, called *binding lists*, exist for the two floorplanning examples shown in Figure 3.

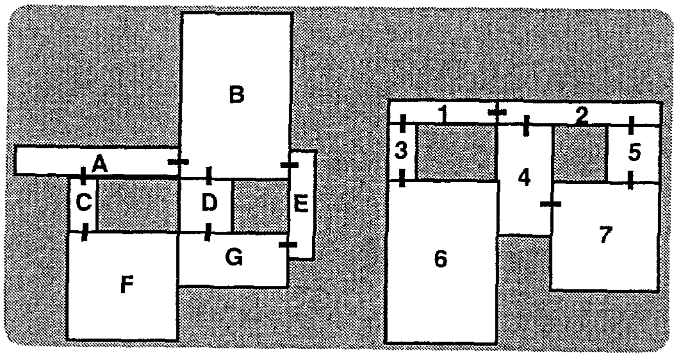

Figure 3: Two simple floorplans that can be matched against each other in *different* ways

Two of them are e.g.
L1: (A:1, B:2, C:3, D:4, E:5, F:6, G:7) or
L2: (A:2, B:6, C:3, D:5, E:1, F:7, G:4)
The first one binds objects according to the structure of the floorplans. Corresponding

[1] We only use one-to-one bindings of objects for our generalization strategy. Many-to-one bindings that are used in logic-oriented systems as KBG (Bisson, 1992) are not feasible for the learning of knowledge about *complex design* tasks like floorplanning. Floorplanning operators like the 'placement of BlockA between BlockB and BlockC' require a definite substructure of related objects to be manipulated. In hypotheses with many-to-one bindings this information is lacking.

blocks have the same relative positions and neighbors in both floorplans. For the second list (L2), the areas and shapes of the blocks determine the bindings. If the main aim of the floorplanning process is to minimize the total connection length, the structure of a floorplan is more relevant than the shape of the blocks. Nevertheless, the areas and shapes must be considered too. Therefore, predicate weights are a suitable mechanism to express the importance of each feature and relation.

In COSIMA selective induction is performed by the multi-staged generalization algorithm[2] that uses these numerical weights intensively. They are used to evaluate each (intermediate or final) generalization. A generalization consists of several facts. Each generalized fact F corresponds to a pair of facts from the two considered examples. The similarity of these two facts delivers a weight[3] for F.

Our calculation of the similarity of two facts is somewhat more simple than the formula used in KBG (Bisson 1992). Let F1 = $p(A_1, A_2, \cdots, A_n)$ and F2 = $p(B_1, B_2, \cdots, B_n)$ be the two facts from the two considered examples E1 and E2 that are generalized into F. F1 and F2 are instances of the same predicate p. The weight of F is defined as the similarity between F1 and F2:

$$\mathrm{sim}(F1, F2) = \mathrm{weight}\ (p) * \prod_{i=1}^{n} \mathrm{sim}(A_i,B_i) \tag{1}$$

The similarity value for any pair of arguments ranges from 0 to 1. The calculation of $\mathrm{sim}(A_i,B_i)$ depends on their type. For two *objects* it is equal to 1, for two *numerical values* it is the quotient of their minimum and maximum[4]. If the difference of the values is above a certain limit, the similarity is too low and the fact is dropped from the generalization. Typically this can take place after several generalization steps.

The weight of a generalization is the sum of the weights of its facts being calculated with Formula (1). This method implies that not necessarily the generalizations with the highest number of facts, but those with many important facts, get the highest ratings. In this way the best description that is used for further generalization steps can be determined heuristically.

The multi-staged generalization splits up each example description into two parts: The *important* part consisting of instances of predicates with high weights and the *additional* part consisting of the other facts[5]. During the first stage only the important example parts are matched. This leads to initial, preliminary MSGs. The second stage matches the additional example parts and completes the initial MSGs adding further generalized facts. Each stage is performed by a SPROUTER-like matching procedure (Hayes-Roth and McDermott, 1977). It searches for the best (initial or final) MSGs by stepwise construction of maximal consistent binding lists.

---

[2] This generalization algorithm was first used in the system LEFT (Herrmann and Beckmann, 1992 and 1994), a predecessor of COSIMA. LEFT has the same selective inductive generalization method but no constructive induction or inductive specialization strategies.

[3] We are calculating weights only for *generalized* facts. The weight of a fact in an example description is identical to the weight of the corresponding predicate.

[4] The range of the numerical arguments in our floorplanning domain is relatively small. It is e.g. [1,20] for the numerical argument of the predicate area. The similarity of two areas is expressed by their ratio. According to our point of view, two blocks with areas 1 and 2 are as similar to each other as two blocks with areas 5 and 10. For *large* numerical domains, a normed similarity measure could be more appropriate (e.g. dividing the difference of two values by the maximum possible value).

[5] Alternatively, the example descriptions can be split into several parts. In this case, for *each* part a separate matching phase must be performed.

<u>Example: Binding of objects after the initial matching</u>

Each of the two simple floorplans in figure 4 has 9 blocks. The description language consists of the three predicates

connected(<block>, <block>)	with the weight 40
size(<block>, <integer>)	with the weight 15
shape(<block>, <size-descriptor>)	with the weight 10

The important part of the example description consists of instances of the first predicate. The additional parts describes the size and shape features. After the initial matching (first stage) only the following blocks are bound to each other:
(A:1, B:2, C:3, D:4, E:5, F:6, G:7)
The other blocks are bound during the second matching stage. The initial bindings are not changes during the second stage.

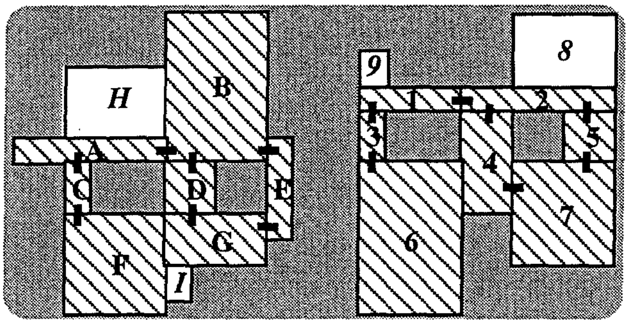

Figure 4: Two simple floorplans after the initial matching: The hatched blocks are already bound to each other

The generalized facts are created by use of the closing interval, climbing generalization tree, turning constants into variables and dropping condition generalization rules (Michalski, 1983). At last the final MSG with the highest rating is selected. A detailed description of the multi-staged generalization algorithm can be found in (Herrmann and Beckmann, 1994). The effects of this algorithm are the following one:
The instance and the hypothesis space are split into 2 disjoint parts. This leads to a significant reduction of the complexity for generalization. On the other hand, few important facts (with high weights) cannot be "outvoted" by many additional (less import) ones. For that reason, our generalization algorithm can find better MSGs for domains like floorplanning that are represented adequately by descriptors with varying importance.

5. Selective Inductive Specialization

The inductive specialization component is applied, if the application of a learned rule, matching on the current floorplanning state, is rejected. The rejection of a rule means that its application would lead to an unfavorable new state according to the floorplanning objectives. The rejection is either initiated by the user or the built-in evaluation function for floorplanning states.
As the floorplanning domain requires an *incremental* learning system (see Section 2), specialization strategies that systematically analyze the whole set of examples at one time cannot be used, e.g. MOBAL's rule discovery tool (Kietz and Wrobel, 1992) or the knowledge refinement component in WHY (Saitta et al, 1993).

One simple method for incremental specialization is the inclusion of exceptional examples into the rule to be specialized, as it is used in the system LEFT (Herrmann and Beckmann, 1994). Using a "without-part" in each rule, negative examples can be excluded easily. Unfortunately this method does not scale up. It blows up the rule description so that the advantages of incremental learning are jeopardized.
An alternative incremental specialization strategy is performed in the system LAIR (Elio and Watanabe, 1991). It compares each new (positive or negative) example with the current hypothesis and uses the differences for the construction and update of a list of candidates for specialization.
LAIR uses a constructive induction method for creation of the fact needed for specialization. It utilizes a set of horn-clauses as background theory. The background knowledge is applied to the candidate list to find the best fact to be added to the rule. Because of the insufficient background knowledge in the floorplanning domain we cannot use this construction method for our system. (COSIMA's alternative constructive induction strategy is explained in the next section.)
Nevertheless a list of candidates can be used for specialization in our domain. The list is constructed and updated in a more distinctive way than in LAIR, which removes a candidate form the list, if it contradicts the current new single example. In COSIMA all facts are stored in the so-called *specialization part* that occurred in at least C % of the examples used for generalization or used during application of a given rule[6]. C is a user selected parameter. Realistic values for C are e.g. 50 or 30. A smaller value for C leads to a bigger specialization part. Each fact in the specialization part is marked with two numeric values:

- The percentage of example descriptions for generalization that comprise the fact
- The percentage of example descriptions the rule has been applied to successfully, i.e. without rejection, that comprise the fact

If a rule matches on a negative example the rule is too general and has to be specialized. Adding a literal to the left-hand side of the rule that *discriminates* the example, this is achieved. In this way a most general specialization (MGS) of the rule is created. In analogy to most specific generalizations (Haussler, 1989) there are many alternative most general specializations, so the selection of the best or at least a good one is crucial. Our specialization strategy uses two different rules for the creation of a MGS (Dietterich and Michalski, 1983), both utilizing the information in the specialization part as bias.

a) The introducing exception specialization rule
The negation of a fact is added to the left-hand side of the rule that occurs in the negative example and in few (at least c %) of the positive ones.

b) The adding condition specialization rule
A fact is added to the left-hand side of the rule that does not occur in the negative example but in most (close to 100%) of the positive ones.

Both rules can lead to a discrimination of old examples that were classified as positive ones. This effect is intended, as the classification of examples is fuzzy in our domain. (There is limited knowledge about the analysis of floorplans, see Section 2.) Therefore, misclassified examples are a major reason for overgeneralizations. Nevertheless, the number of discriminated old examples must be kept small, to minimize the effect of the specialization. The specialization part provides the necessary information to meet this

[6] To make the construction of the list less sensitive to the presentation order of the examples, for the fist examples *all* possible candidate facts are included. Only after a number of examples have been used for generalization or application of the rule, the less frequently occuring facts are discharged.

requirement. Other incremental specializations strategies, like e.g. the one in ACT (Anderson, 1986), have a less elaborated discrimination mechanism. The discrimination is based only on the analysis of one positive and one negative example. The influence on the total set of old examples cannot be estimated.
There are several criteria for the evaluation of the alternative specializations created with rule a) and b). The selection of the "best" literal to be added to the rule is based on the following information:

- *Number of positive examples discriminated by the literal*
  As has been mentioned above this number should be minimized to limit the effect of specialization
- *Predicate weight for the instance*
  In our current implementation predicates with high weights are preferred, as they add significant information to the hypothesis
- *Statistics about prior use of the corresponding predicate for discrimination*
  Statistics represent the success of predicates during previous specialization steps. If a predicate has lead to a successful specialization several times, it is a good candidate for the current specialization step according to this heuristic criterion.
- *Number of objects referenced in the rule before and after specialization*
  The addition of a single literal to a hypothesis can increase the number of referenced objects. This is a significant modification of the hypothesis. Therefore, specialization steps that do not increase the number of objects are preferred in COSIMA.

These four criteria provide basic information about the evaluation of the specializations. The way how they should be combined to gain a compound evaluation function depends strongly on the characteristics of the underlying domain.

Besides the rules a) and b) that add a literal to the rule to be specialized COSIMA uses the following specialization rules that are the inverse operations to standard rules of generalization: descending-concept-hierarchy-tree, reducing-interval, turning-variable-into-constant, removing-alternative. A formal description of the corresponding generalization operators can be found in (Michalski, 1983).
Each of these four rules specializes an argument of a fact already existing in the considered floorplanning rule. As the influence of these specializations on the total set of old examples cannot be estimated, the rules a) and b) are the preferred specialization operators in COSIMA.

6. Three Different Ways to Use Constructive Induction

COSIMA's constructive induction strategy combines knowledge-based and syntactical construction of new descriptors. Knowledge-based approaches, e.g. Oxgate (Gunsch, 1991), use domain-dependent knowledge for the construction process. In contrary, syntactical approaches (Wirth and O'Rourke, 1991) are based on domain-independent biases limiting the space of predicates that can be constructed.
In the following the three different uses for constructive induction in COSIMA are explained sequentially.

Modification of Example and Rule Descriptions Before Matching

One purpose of the modification (and the general intention of constructive induction) is to make the descriptions more distinctive, i.e. to make information explicit that is only implicitly represented in the descriptions. For this purpose COSIMA's background knowledge provides several predefined construction operators. They create new predicates, being possibly useful for our domain, from existing "basic" ones. Examples of operators are e.g. the following two ones (Michalski 1983):

- *Counting the instances of a predicate that all have the same constant value at one argument position*
  Example: Consider the following basic predicate:
  block_state(<block>, <state-value>). The possible values for <state-value> are `placed` and `unplaced`. The constructed predicate
  #placed_blocks(<number>) makes explicit the number of blocks with the state-value `placed`.
- *Selecting that instance of a predicate with the maximum numerical value at one argument position*
  Example: The constructed predicate maximum_size(<block>) makes explicit that the object <block> is the biggest one.

Each of the construction operators can create a diverse set of different predicates. For instance the maximum-operator can be applied to any predicate with at least one numerical argument type. Some of the operators depend on each other. For the creation of a maximum-predicate a counting-instances-predicate must be used.
The constructed predicates that get a sufficient rating from the predicate evaluation function (see below) are included into the example and the considered rule description. It would not be feasible to add the predicate to the description language for all rules. This would blow the descriptions significantly. Instead, the effect of the construction is limited to the current rule. For different rules different constructed predicates can be relevant.
The second type of modification for example and rule descriptions is the compaction. For this purpose intermediate concepts represented as horn-clauses are induced (Bergadano et al, 1988). An intermediate concept represents a compound property occuring repeatedly in the set of examples.
If the body of a horn-clause (with two ore more literals) matches on an example the unified literals in the example description are substituted with the clause head. (In the system DUCE (Muggleton 1987), this operation, called absorption if the horn-clause is predefined, was used for transformation of propositional descriptions.) Using intermetiate concepts the size of the description is decreased and high-level information about the example is extracted. The left-hand side of Figure 5 shows an example of an intermediate concept used in COSIMA. The right-hand side depicts a geometrical illustration of the concept "corner". A corner consists of three connected blocks that form an "L-shaped" structure. Using intermediate concepts of this kind, several blocks can be combined to substructures of the floorplan topology.

```
corner(a,b,c) :-  on_same_line(a,b,horizontal),
                  on_same_line(a,c,vertical),
                  direct_neighbour(a,b),
                  direct_neighbour(a,c),
                  state_connection(a,b,existing),
                  state_connection(a,c,existing)
```

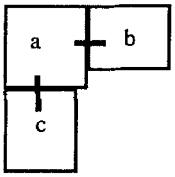

Figure 5: Intermediate concept representing the structure "corner" (left-hand side) and the geometrical representation of a corner in a floorplan example (right-hand side)

COSIMA uses rule-models (Kietz and Wrobel, 1991; Pazzani and Kibler, 1992) for syntactical construction of predicates representing intermediate concepts. The head of a successfully instanciated model forms a new predicate to be included into the considered description. To limit the set of possible new predicates that can be constructed from a rule model, *sorts* are used constraining the instanciations of predicate variables and arguments. In Figure 6 the rule model is depicted that was used to create the intermediate concept for "corner".

```
new(X,Y,Z) :-        P(X,Y,horizontal) ∧ Type (P) = IndirectNeighbourhood,
                     P(X,Z,vertical),
                     Q(X,Y) ∧ Type (Q) = DirectNeighbourhood,
                     Q(X,Z),
                     R(X,Y,existing) ∧ Type (R) = Connection,
                     R(X,Z,existing)
```

Figure 6: Rule model in COSIMA. "new" stands for a predicate identifier to be created automatically during instanciation of the model. (Meaningful predicate names have to be inserted by the user.) The italic terms denote the sort restrictions.

COSIMA's two different construction methods can be combined. A new predicate constructed from a *predefined construction operator* can be included into the body of an *intermediate concept.* Besides that, intermediate concepts can be created hierarchically: More complex intermediate concepts are defined by use of existing ones.
To make the creation of intermediate concepts less dependent on the existing rule models, COSIMA has a method for the creation of new models based on past successful instanciations of existing ones. If there are several instanciations of the same model, a new clause is created with a body consisting of the corresponding literals in all instanciations. A new rule model is abstracted from this clause. For a more detailed description of the constructive induction mechanisms see (Reipa, 1993).

Evaluation of the Constructed Predicates

This is a crucial task for any constructive induction strategy. Typically, a flexible construction mechanism results in a big number of *irrelevant* new predicates. COSIMA's evaluation function is based on the following criteria:

- number of literals in the rule model used
- number of instances of the new predicate in the two considered descriptions
- degree of compaction achieved with the new predicate
- past success of the rule model

From these criteria a heuristic numerical value is calculated for each new predicate. Only the predicates with a high rating are included into example and rule descriptions.

Selection Among Alternative MSGs

This is the second use for constructive induction in COSIMA. From structural description many different alternative most specific generalizations (MSGs) can be created. The *weight* of each MSG is the sum of the weights for the facts. If there are several MSGs with the same highest rating, constructive induction is used to select the best one.
If applicable, COSIMA adds instances of constructed predicates to an MSG making it more specific in that way. A fact is added, if it is valid for the corresponding objects in all examples[7]. This increases the weights for some MSGs and makes a selection possible.

[7] This condition already limits the number of accepted constructed predicates significantly. For this reason the threshold value for the acceptance of new predicates, that reduces this number further, is now lower than during the modification of descriptions (see above).

During this stage of the generalization process new facts are only *added* to the MSGs, no compaction takes place.

Specialization of Rules

If the inductive specialization strategy mentioned above fails to select a discriminating fact, constructive induction is used as an alternative specialization strategy. A new fact is constructed that discriminates the negative example from the positive ones.

COSIMA's constructive induction strategy is incremental. For the creation of a new predicate the two current descriptions (rule and example) are analyzed. When a new positive example becomes available and is generalized with the rule the utility of the new instances is checked. If they are not valid for the new example, they are dropped.

7. Adjustment of Parameter Weights

Experts know what information is absolutely necessary for a certain operation in the considered domain. They can therefore divide the set of predicates into the two required subsets (important predicates and additional ones) and deliver possibly a partial ordering for each set. From this information initial predicate weights can be determined. A modification of these weights is performed by a statistical parameter adjustment mechanism. Analyzing the generalization and specialization steps regularly it evaluates the relevance of the different predicates for the learning process and adjusts the weights incrementally.

8. Representation of Floorplanning Knowledge in COSIMA

Floorplanning may be divided into two steps: The *topology planning* which performs a rough, relative placement of the blocks, and the *geometry planning* which determines the exact floorplan geometry based on this topology.
COSIMA incrementally acquires rules which create a floorplanning *topology*.[8] The topology is represented as a *grid-graph* (Watanabe, 1987), a graph with (square) nodes which are marked with positions on a two-dimensional grid. The nodes represent the floorplan blocks; grid-positions stand for the relative placement of the blocks. Two blocks are connected by an edge if there is a corresponding connection in the circuit description COSIMA gets as an input. Four rectangles represent the boundary of the floorplan. The upper left part of Figure 7 show a grid graph with some blocks that have already been placed.
A logic-based description is not well suited for the representation and manipulation of structural and geometrical knowledge. For this reason we use a hybrid representation for COSIMA. The manipulation of the current floorplanning state is performed by a set of predefined floorplanning operators that operate on an object-oriented representation, implemented in the CommonLisp Object System CLOS. Each operator is implemented using quite a complex CommonLisp procedure. After each manipulation, the new state is translated into the predicate description which the learning component can work on.

[8] The geometry planning is more straight-forward and can be calculated by a conventional algorithm (Watanabe, 1987).

9. Integration of the Learning System into a Floorplanning Tool

The learning strategies described above are embedded in an interactive floorplanning system that assists the designer. LEFT has a graphical interface that shows the current floorplanning state and which is used by the designer to initiate the application of an operator (see Figure 7).

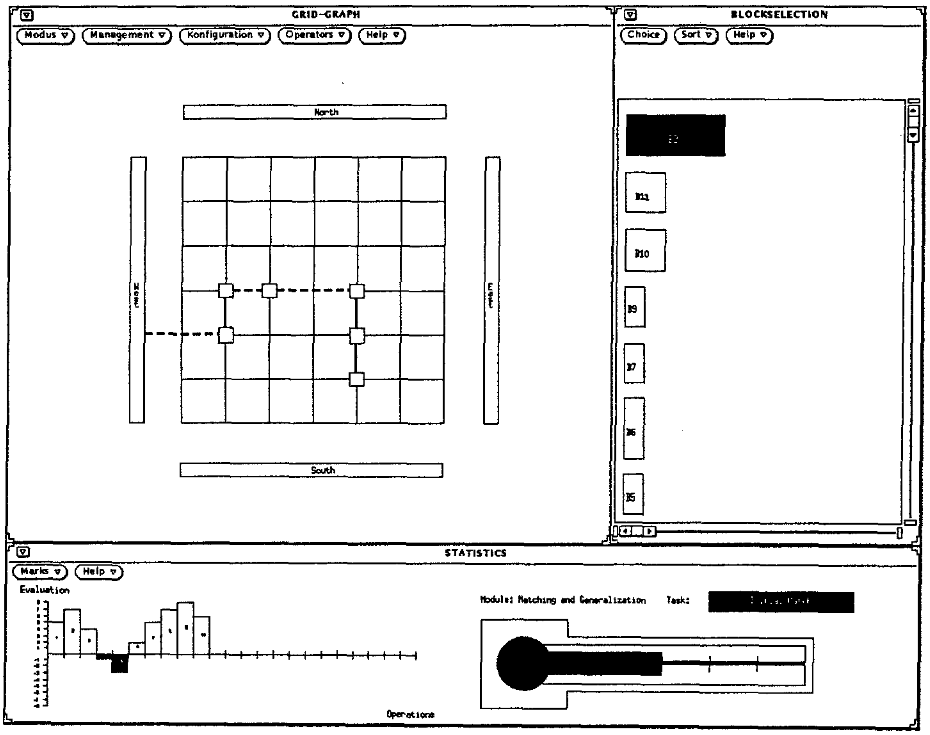

Figure 7: User Interface of COSIMA. The upper right window shows the ordered list of blocks to be placed. The blocks are placed onto the positions of the grid shown in the upper left window. In this way the topology of the floorplan is designed. The lower window depicts performance statistics. The graph on the left-hand side shows the quality of the most recent floorplanning operations. The user can take back unfavorable operations. The bar on the right hand side informs about the state of current matching and generalization process.

The system operates as a learning apprentice (Mitchell et al, 1985). The user can select from the list of possible operations an appropriate floorplanning operator for the next design step. The performance component executes the operator. This leads to a new floorplanning state. The old state description and the operator form a positive example. It is used to construct a new rule or to generalize an existing one. The learned rules are used in the following way: To make the selection of a well-suited operator easier, each time a rule matches on the current state the execution of the operator in the right-hand side is proposed to the user. If the user rejects the proposed operator the current state forms a negative example for that rule and it is specialized accordingly.

10. Experimental Results

We are currently evaluating the quality and capability of COSIMA's different strategies. This includes for instance the selection of appropriate parameters for the different evaluation functions.
The effect of the multi-staged generalization algorithm on the speed of matching and generalization has already been evaluated and is illustrated in Figure 8. The algorithm is compared to another version that is limited to a single stage, i.e. all predicates are considered during one single generalization stage. Both versions used the same predicate weights. The results show that the multi-staged generalization algorithm can significantly improve the run time for inductive generalization taken from real-world design examples. (The single-staged generalization is only quicker for very small examples.)
On top of which, the tests show that the multi-staged generalization improves the quality of the gained learning result. It selects the appropriate MSGs for several test cases, which could not be generalized correctly by the single-staged generalization algorithm.

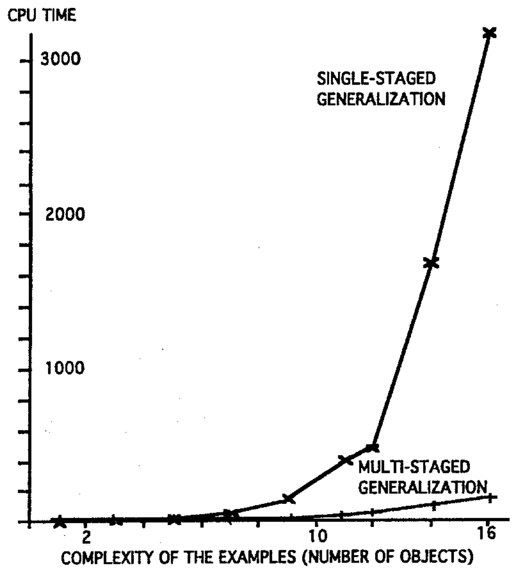

Figure 8: Comparison of Run-Times for the Single- and the Multi-Staged Generalization

There are some preliminary results about the effects of the other machine learning strategies, based on a limited number of tests we have already performed[9]. Figure 9 depicts the results of one test series. In each of the floorplan examples used for this series at least one corner occurred (see Figure 5 for a description of the intermediate concept corner). The example size varied from 250 to 520 facts.
We have investigated how the description length of a rule created from these examples changed during multi-staged generalization with and without constructive induction. It is the typical case for realistic floorplanning examples that only a smaller part of the example description is significant for the concept to be learned. Therefore, the dropping condition generalization rule is the most important one.

[9] Therefore, at the moment our interpretation of the experimental results is somewhat fuzzy.

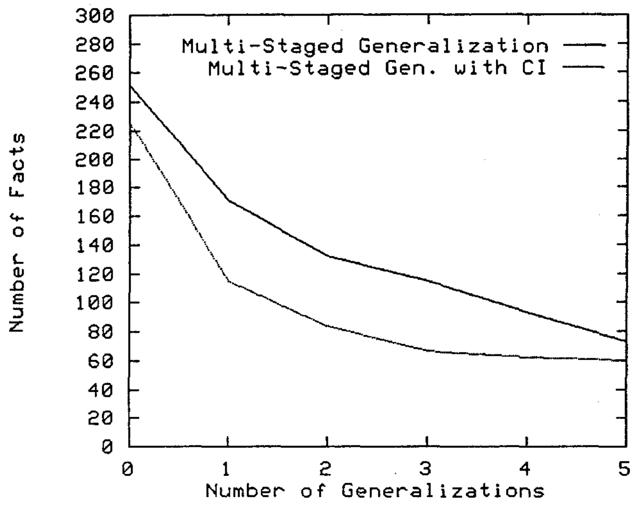

Figure 9: Comparison of rule sizes for multi-staged generalization without constructive induction (upper line) and with constructive induction (lower line)

The combination of constructive induction with multi-staged generalization had several effects. From the beginning it decreased the rule size, and it lead to an earlier convergence of the number of facts. This combination of the two machine learning strategies does also influence the quality oft the learned rules, as has been confirmed by several other series of tests, too. It decreases the error rate of the rules significantly.

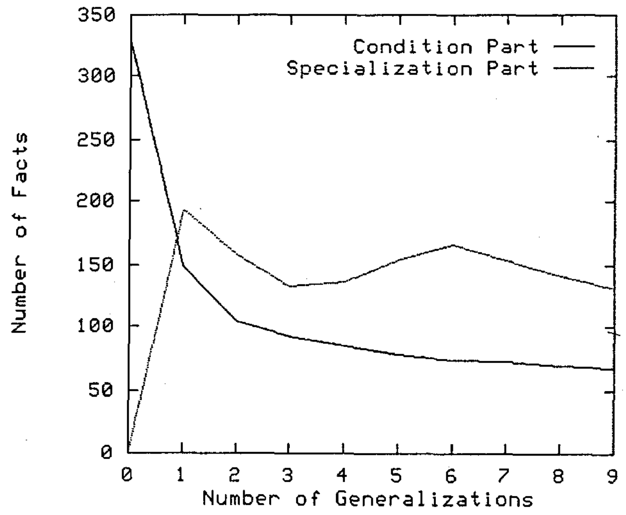

Figure 10: Comparison of number of facts for the specialization part and the condition part of a rule that is generalized several times

Another series of tests analyzed the development of the specialization part during a number

of successive generalizations (see Figure 10). In this series the size of the specialization part was always somewhat larger than the size of the condition part, but the specialization part did not grow significantly during generalization. There is no combinatorial explosion of the specialization part. This is an important difference to the simple specialization strategy of the system LEFT (Herrmann and Beckmann, 1994) that blew up the size of the specialization part after learning from a few negative examples.
Another aspect that has still to be analyzed and is relevant for all incremental systems is the influence of the example order on the learning results.

11. Conclusion

The implemented system COSIMA combines different machine learning strategies. It demonstrates how selective inductive generalization, inductive specialization, constructive induction and statistical adjustment of parameter weights can be integrated into a multi-strategy machine learning system that acquires knowledge about a real-world problem - floorplanning for integrated circuits.
Another important aspect of machine learning applications is the integration of a learning system into the daily work of the user. The organization of the user interaction has a great impact on the acceptance for the software tool. Considering this aspect, COSIMA has been organized as a learning apprentice system that is integrated into an interactive floor-planning tool. COSIMA does not require a teaching mode. It learns from examples that the user generates during his/her normal work with the floorplanning tool.

Acknowledgements: We would like to thank Siegfried Bell for useful comments on an earlier version of this paper.

References

Anderson, J.R. (1986). Knowledge Compilation. In Machine Learning: An Artificial Intelligence Approach Vol II, eds. R. S. Michalski, J. G. Carbonell, T. M. Mitchell, 289-310. Morgan Kaufmann..

Bergadano, F., Giordana, A. & Saitta, L. (1988). Automated Concept Acquisition in Noisy Domains. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Bisson, G. (1992). Conceptual Clustering in a First Order Logic Representation. Proc. 10th ECAI, August 3-7, Vienna.

Dietterich, T.G., & Michalski, R.S. (1983). A Comparative Review of Selected Methods for Learning from Examples. In Machine Learning: An Artificial Intelligence Approach, eds. R. S. Michalski, J. G. Carbonell, T. M. Mitchell, pp 41-81. Palo Alto: Tioga Press.

Elio, R., & Watanabe, L. (1991). An Incremental Deductive Strategy for Controlling Constructive Induction in Learning from Examples. Machine Learning Journal. Vol 7, 7-44, Boston: Kluver Academic Publishers.

Gunsch, G.H. (1991). Opportunistic Constructive Induction: Using Fragments of Domain Knowledge to Guide Construction. PhD Thesis, University of Illinois at Urbana-Champaign.

Haussler, D. (1989). Learning Conjunctive Concepts in Structural Domains. Machine Learning Journal. Vol 4, 7-40, Boston: Kluver Academic Publishers.

Hayes-Roth, F., & McDermott, J. (1977). Knowledge Acquisition from Structural Descriptions. Proc. 5th IJCAI, 356-362.

Herrmann, J., & Beckmann, R. (1992). LEFT - A Learning Tool for Early Floorplanning. Proc. 18th Euromicro Conference, pp 587-594, September 14-17, Paris.

Herrmann, J., & Beckmann, R. (1994). LEFT - A System that Learns Rules about VLSI-Design from Structural Descriptions. (to appear) In Y. Kodratoff (Guest Ed.), Applied Artificial Intelligence, Special Issue on Real-World Applications of Machine Learning Techniques . London: Taylor and Francis Ltd.

Kietz, J.U., & Wrobel, S. (1992). Controlling the Complexity of Learning in Logic Though Syntactic and Task-Oriented Models. Arbeitspapiere der GMD Nr. 503, GMD, Schloß Birlinghoven.

Kodratoff, Y., & Langley, P. (Eds.), (1993). Real-World Applications of Machine Learning. Workshop Notes on the ECML-93 Workshop. Vienna.

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In R.S. Michalski, T.M. Mitchell and J.G. Carbonell (eds.), Machine Learning: An Artificial Intelligence Approach. Palo Alto, CA: Tioga Publishing.

Michalski, R.S. (1993). Inferential Theory of Learning as a Conceptual Basis for Multistrategy Learning. Machine Learning Journal. Vol 11, 111-152, Boston: Kluver Academic Publishers.

Mitchell, T.M., Mahadevan, S., & Steinberg, L. (1985). LEAP - A Learning Apprentice for VLSI Design. Proc. 9th IJCAI, pp 573-580, August 18-23, Los Angeles.

Morik, K. (1992). Applications of Machine Learning. Proceedings of the Sixth European Knowledge Acquisition Workshop (pp 9-13). Springer.

Morik, K. (1993). Balanced Cooperative Modeling. Machine Learning Journal. Vol 11, 217-236, Boston: Kluver Academic Publishers.

S. Muggleton (1987). DUCE, an Oracle Based Approach to Constructive Induction, Proc of the 10th Int. Joint Conference on Artificial Intelligence, IJCAI87

Pazzani, M., & Kibler, D. (1992). The Utility of Knowledge in Inductive Learning. Machine Learning Journal. Vol 9, 57-95, Boston: Kluver Academic Publishers.

Reipa, D. (1993). Konstruktive Induktion für eine strukturelle Beschreibungssprache. Diploma Thesis, University of Dortmund.

Saitta, L., Botta, M., & Neri, F. (1993). Multistrategy Learning and Theory Revision. Machine Learning Journal. Vol 11, 153-172, Boston: Kluver Academic Publishers.

van Someren, M. (Ed.) (1993). Learning and Problem Solving. Workshop Notes on the MLnet Workshop. Blanes.

Wirth, R., & O'Rourke, P.O. (1991). Constraints for Predicate Invention. Proc. of the 8th Int. Machine Learning Conference, Evanston: Morgan Kaufmann.

Watanabe, H. (1987). FLUTE - An Expert Floorplanner for Full-Custom VLSI Design. IEEE Design & Test, pp 32-41. New York: Computer Society of the IEEE.