Concept Sharing: A Means to Improve Multi-Concept Learning

Piew Datta
Information and Computer Science Dept.
University of California, Irvine
Irvine, CA 92717-3425
pdatta@ics.uci.edu

Dennis Kibler
Information and Computer Science Dept.
University of California, Irvine
Irvine, CA 92717-3425
kibler@ics.uci.edu

Abstract

This paper describes several means for sharing between related concepts to improve learning in the same domain. The sharing comes in the form of substructures or possibly entire structures of previous concepts which may aid in learning other concepts. These substructures highlight useful information in the domain. Using two domains, we evaluate the effectiveness of concept sharing with respect to accuracy, concept size, search complexity, and noise resistance.

1 Introduction

Humans ability to learn is remarkable. One characteristic of human learning is that it employs multiple sources of knowledge, including examples, partial theories, and background knowledge. Markman (Markman, 1989) argues that human learning has a number of inherent biases which aid in the learning of natural concepts. We seek computational biases that can increase the accuracy of learned concepts.

We take a broader view of concept learning, namely that of learning multiple concepts instead of learning single concepts in a domain. A multi-class decision tree also learns multiple concepts, however, it does not employ our bias of concept sharing. When learning a set of concepts, concepts may share common substructures. Concept sharing allows partial or complete information from previously learned concepts to be shared for future learning. Our premise is that concepts from the same domain share some content, and therefore, substructures that are frequently used for learning concepts previously will be applicable for learning concepts subsequently. Concept sharing biases the representation of concepts to prefer sharing common substructures. We feel that sharing concept information is an additional form of knowledge that has not yet been exploited.

Various means for sharing structures and substructures among concepts exist. Sharing an entire concept structure among concepts is similar to systems that perform analogy and case-based reasoning. Their assumption about the similarity of cases and domains parallels our premise about sharing concept information. Sharing a substructure among concepts emphasizes useful pieces of information within a domain. This type of sharing is similar to constructive inductive systems that create substructures by applying operators on the primitive description elements (Schlimmer, 1987; Pagallo & Haussler, 1990).

Our goals for this paper are two-fold. First, we will introduce the problem of learning multiple concepts for inductive learning systems. Secondly, we demonstrate reuse of knowledge from prior concepts for learning subsequent concepts. An illustration of this information transference is implemented in Multiple concept FOCL (M-FOCL). M-FOCL is a relational concept learning algorithm that learns multiple concepts sequentially, while sharing knowledge among concepts. In Section 2, we review two relational learning algorithms FOIL and FOCL, on which an analysis of M-FOCL is based. Section 3 explains M-FOCL in greater detail and section 4 evaluates the effectiveness of the concept sharing by testing M-FOCL on two domains: chess and poker. We contrast our work with related work in learning multiple concepts in section 5. We conclude by summarizing our contributions and discussing future work.

2 Relational Horn clause learning algorithms

Before describing Multiple concept FOCL (M-FOCL) and FOCL (Pazzani & Kibler, 1990), we will describe FOIL (Quinlan, 1990) to explain the core learning method applied in these systems. Given a set of extensional predicates, positive examples, and negative examples, FOIL learns a relational Horn clause concept in terms of predefined predicates. This concept covers all of the positive examples and excludes all neg-

ative ones. FOIL constructs the concept by performing a hill-climbing search. It begins creating a clause by finding the predicate (including the choice of variables) that has the highest information gain. This predicate will cover some positive examples and possibly cover some negative examples. FOIL specializes the clause to exclude negative examples by conjoining additional predicates. After constructing a clause, FOIL discards all the covered examples and repeats the same process until no positive examples remain uncovered. The final concept is a disjunction of clauses.

FOCL follows FOIL's method for constructing concepts, but FOCL utilizes intensional predicates in the form of rules. These predicates are conjunctions or disjunctions of other predicates. They allow the user to define an existing domain theory for the relational learning system. FOCL's ability to use a domain theory allows it to reuse its own output, a Horn clause description, as input for learning another concept.

3 Multiple concept FOCL

The premise underlying the design of M-FOCL is that different concepts in the same domain share common content, and thus information gained from learning a concept is transferred when learning other concepts. When learning multiple concepts, M-FOCL adds the learned concepts to its domain knowledge. According to our premise these additional concepts should facilitate learning subsequent concepts. This allows M-FOCL to learn the concept and the types of information that are important in the domain. Thus by biasing the algorithm towards previously learned information, an increase in accuracy and efficiency may be possible.

The inputs to M-FOCL are positive and negative training examples, primitive predicates, and predicates of previously learned concepts. As output M-FOCL returns a relational Horn clause concept covering all positive examples and no negative examples, and adds the concept just learned to the predicates. The updated list of predicates is input back into M-FOCL when learning another concept, so that reuse of previous information is available. M-FOCL differs from FOCL in that it increases the available intensional predicates by creating predicates from concepts based on their past usefulness.

Currently M-FOCL takes advantage of the content of previously learned concepts in three different ways. Suppose that M-FOCL has learned the concept C, defined in pseudo-Prolog (Figure 1). We use C_i to represent clauses and P_{ij} to represent predicates. To simplify the description, we have not included the variables associated with C, C_i, or P_{ij}.

The first method, the *concept rule*, adds C as an intensional predicate to the domain knowledge. C is a

$$
\begin{aligned}
C &: -P_{11}, P_{12}, \ldots, P_{1n_1} \equiv C_1 \\
C &: -P_{21}, P_{22}, \ldots, P_{2n_2} \equiv C_2 \\
&\vdots \\
C &: -P_{k1}, P_{k2}, \ldots, P_{kn_k} \equiv C_k
\end{aligned}
$$

Figure 1: Concept C

disjunction of all clauses C_1 through C_k. The second method, the *clause rule*, adds a clause, C_i, for some i, to the language as an intensional predicate. The third method, the *conjunct rule*, adds a conjunction of predicates as an intensional predicate to the language. The conjunction consists of at least two predicates, P_{ix} and P_{iy}, that are also used in another clause C_j where $j \neq i$, and Cj can be from the concept just learned or from a background theory of past concepts. We only consider conjunctions that appear in more than one clause to reduce the number of new predicates introduced into the representation language. M-FOCL adds these predicates to its primitive predicate set when learning concepts. The applicability of these predicates for learning subsequent concepts will be determined by the information gain heuristic used in FOCL. The single difference between FOCL and M-FOCL is that M-FOCL uses these predicates during induction, whereas FOCL does not create these predicates and thus cannot use them. In a sense, M-FOCL is dynamically changing its representational bias by introducing new terms that have been found useful in the past.

The concept rule is useful in situations where the concept to be learned is a generalization of a previous concept. For example, when learning legal chess moves, a queen move subsumes a rook move, that is, by applying the concept rule, M-FOCL uses a rook move as part of the queen move definition. The clause rule creates a predicate from a clause that is useful when concepts share partial information or a concept is a specialization of a previous concept. For instance, if a queen move is already defined, when learning a rook move, M-FOCL uses one of the clauses defined in the queen concept that allows the queen to move vertically on the board, as part of the definition of the rook. Lastly, the conjunct rule is useful when concepts share some content, but do not share entire concepts or clauses. For example, since each piece must be on the board, each clause must represent this constraint. This constraint is not represented as an entire concept or a clause, but as a conjunction of some predicates. By using the conjunct rule, a predicate is created that is the conjunction of other predicates that form useful partial substructures.

An analogy can be drawn between learning new predicates in M-FOCL and learning macros in planning systems. Macros are learned from previous successful sequences of operators (Ruby & Kibler, 1991), whereas the concept, clause, and conjunct rules facilitate learn-

ing new predicates from previous concepts. In the same sense that macros aid the planning system to take larger steps in the hill-climbing search, creating new predicates helps M-FOCL take larger steps in its hill-climbing search, thereby changing the landscape of the hill-climbing search as well as potentially increasing the learning rate. By changing the landscape of the hill-climbing search, misleading local optima points may be avoided.

Using these three rules also allows M-FOCL to do less search for relevant predicates than FOCL. The effect of using previously learned substructures is to do selective lookahead in the hill-climbing search. If the selection is good, improved performance will result. Creating intensional predicates often reduces the depth of the search when a good selection is chosen, but it also increases the branching factor of the search. Therefore, the trade-offs for generating intensional predicates with these rules must be considered. This is analogous to the utility problem (Minton & Carbonell, 1987) while learning macros for planning.

In accordance with the idea of concept sharing, we developed the concept, clause, and conjunct rules. These rules facilitate M-FOCL's use of prior concepts to learn subsequent concepts. The examples that were used to create concepts are indirectly used to learn other concepts. When learning with noisy data, indirectly utilizing past examples has the effect of generating more examples to learn the concept and isolate noisy examples. They also may decrease search since M-FOCL uses previously learned information.

4 Evaluation

To empirically evaluate the usefulness of concept sharing, we tested FOCL against M-FOCL in two domains for our preliminary studies. The remainder of this section describes those domains in greater detail, describes the methodology used to run the experiments, and explains the experimental results.

4.1 The domains

The two domains used to evaluate M-FOCL are that of learning chess moves and learning poker hands. The learning problem in the chess domain is similar to those used in Vere (1977) and De Raedt & Bruynooghe (1992), however our experiments test a subset of the chess moves, specifically legal moves for the rook, bishop, knight, and queen. The moves are learned sequentially. Each example consists of the initial position of the piece and its final position described by the 4-tuple *(initial-rank,initial-file,final-rank,final-file)*. The moves are learned as if no other pieces are on the board.

The learning problem in the second domain consists of learning a subset of the possible five card poker hands. Specifically, the following concepts are learned: pair, three of a kind, four of a kind, two pair, and full house. For example, a *pair-in-hand* is defined as a hand with at least two cards of the same rank. The examples given to the system consist of the 5-tuple *(card1, card2, card3, card4, card5)* where the primitive predicates *rank(card,r)* and *suit(card,s)* describe the rank and suit of each of the cards.

Another representation we considered used the 1-tuple *(hand)* to describe the examples. The primitive predicates for this representation contained *card-in-hand(hand, card), rank(card,r),* and *suit(card,s)*. Thus the only difference between this representation and the former is that the former representation describes a relationship among the objects (cards), whereas in the latter representation, the concept is described using one object (hand) which consists of several objects (cards). The latter representation seems more intuitive, since it describes in detail the highly relational characteristics of the domain; however, FOCL and M-FOCL have space limitation problems when learning the concepts with this representation. For example, suppose FOCL is learning *pair-in-hand(?hand)* (where ? symbols preceding words denote variables) and it has already learned *card-in-hand(?hand, ?card1)* as the first literal in a clause. Since each hand contains five different cards, the number of possible bindings for the unbound variable *?card1* greatly increases the search space of possible horn clause theories (Quinlan, 1990). Indeed, the search space increases by a factor of 5, the number of cards in the hand. Next FOCL would add *card-in-hand(?hand,?card2)* which increases the search space by another factor of 5. In fact, each time FOCL adds the predicate *card-in-hand* the search space increases by a factor of 5. This representation may be reasonable when learning *pair-in-hand* since we need only two literals of the card-in-hand predicate, but FOCL quickly runs into space limitations when learning the other concepts such as three of a kind or full house, that require more instantiations of the *card-in-hand* predicate.

Although using the former representation with the 5-tuple *(card1, card2, card3, card4, card5)* avoids the extended tuple explosion, it suffers from a related search explosion. FOCL learns all of the concepts in the poker domain, but when M-FOCL learns one concept it uses the concept rule to change the concept into a rule for later use. The problem occurs when trying to evaluate the usefulness of the 5-arity concept rules. For example, suppose M-FOCL has already learned *pair-in-hand(?card1, ?card2, ?card3, ?card4, ?card5)* and will use that concept when learning *three-of-a-kind-in-hand*. When M-FOCL checks the information gain of *pair-in-hand* it must try the 120 combinations of the five variables when in reality the ordering of the variables does not affect the information gain. Our solution to this problem was to apply a commutative prop-

erty on the variables in the predicate, allowing different orderings of the variables to be considered the same. We used the commutative constraint on the predicates (Pazzani & Kibler, 1992) and extended it from working only on 2-arity predicates to n-arity predicates. This constraint reduced the amount of search, but was only originally applied to pairs of variables. By extending this constraint to n-arity predicates, the search of each of the concept rules was reduced from 120 variable orderings to one ordering. This technique of using commutativity can be used whenever there is a relation among a set of objects where the ordering of the objects is irrelevant.

4.2 Methodology

To evaluate the usefulness of transferring information among concepts, we tested M-FOCL running in three different modes against FOCL. By running each of the modes separately, each of their effects on learning can be observed independently. In the first mode M-FOCL only used the concept rules (CNR) to create predicates; in the second mode, it uses only the clause rules (CLR) and in the last mode it only uses conjunct rules (CJR). FOCL was run without using any domain information. The tests were run in the following manner. When FOCL was used, a concept from the domain was learned, relevant information such as accuracy was recorded, and then the concept information was discarded. When M-FOCL was run in any mode, no initial theory was used. M-FOCL would first learn a concept. We then recorded the same information as that for FOCL; however, instead of discarding the concept, M-FOCL uses previously learned substructures for learning subsequent concepts.

We ran two experiments in the chess domain, one without any type of noise and the second with attribute noise. We also ran one experiment in the poker domain without noise. In all of the experiments we will report on the average accuracy and search complexity of the algorithms. Their accuracy is measured by using the final concept learned from each of the algorithms to classify randomly generated test examples. The percentage of these examples classified correctly is a measure of the accuracy of the algorithm. The search complexity of the algorithm is the number of predicates and variables tested to learn a concept.

In addition to accuracy and search complexity, we also measured the size of the learned concept. Since the M-FOCL methods create a 'background' theory by applying its three rules, we cannot just count the number of literals in each of the concepts. We took the sum of the literals needed to learn each of the concepts and divided by the number of concepts to get an average concept size. We used the method in Figure 2 to compute the concept size which measures concept and theory literals.

```
for each concept
  for each clause in concept
    for each literal in clause
      if primitive-literal
         or concept-rule
         or conjunct-rule
         or clause-rule then
          add 1 to total-literals
      if (conjunct-rule or clause-rule)
         and not used before then
          add number of rule body literals
              to total-literals
```

Figure 2: Measurement method for concept size

4.3 Experimental evidence

4.3.1 Learning noise-free chess moves

In this experiment we tested FOCL against M-FOCL running in three modes (CNR, CLR, CJR) in the chess domain. All of the chess examples were noise-free, but we varied the training size to see their effects on system search complexity, concept accuracy, and concept description size. According to our premise, M-FOCL should be able to use information from some concepts to aid learning other concepts. We used training sizes of 5, 15, 25, 35, and 45 of each positive and negative examples for each concept. We began with a set of five examples, used those to learn the concepts, and then added 10 more examples to create the training size of 15. Thus each example set was built on the previous set, resulting in a learning curve for the training set. For all training sizes, we evaluated the learned concept with 200 randomly generated positive and negative examples each. This experiment was run with 30 trials where one trial consists of learning all of the concepts in the domain using FOCL and the three M-FOCL methods on a random set of examples. We used the ordering for the M-FOCL methods, rook, bishop, queen, knight to demonstrate the advantage of concept sharing. Previously, we experimented with all orderings which when averaged together had similar results as with this ordering.

We averaged the accuracy over the 30 trials and each of the concepts. The M-FOCL methods were up to 3% more accurate on average than FOCL [1]. The CNR and CLR methods had the highest difference in accuracy from FOCL, whereas the CJR methods accuracy was the closest to FOCL's accuracy. The CJR method's behavior was close to FOCL's because there are only a few sets of substructures that are shared among con-

[1] M-FOCL using the CNR and CLR modes is statistically different than FOCL with 99% confidence for all training sizes. The CJR mode is statistically different than FOCL with 95% confidence for training size 35 and 99% confidence for training size 45. All significance tests use a one-sided t-test.

cepts in this domain, thus few conjunct rules were created. In terms of search complexity, the CNR method searched a space from .2% to 14% smaller than FOCL, and also had a higher accuracy than FOCL on all training sizes. However, some of the M-FOCL methods searched a larger space than FOCL. The CLR method searched a space 26% larger than FOCL for training size 5, but that difference was reduced to 4.2% as the training size increased to 45. The CJR method searched a space almost the same size as FOCL (Figure 3a).

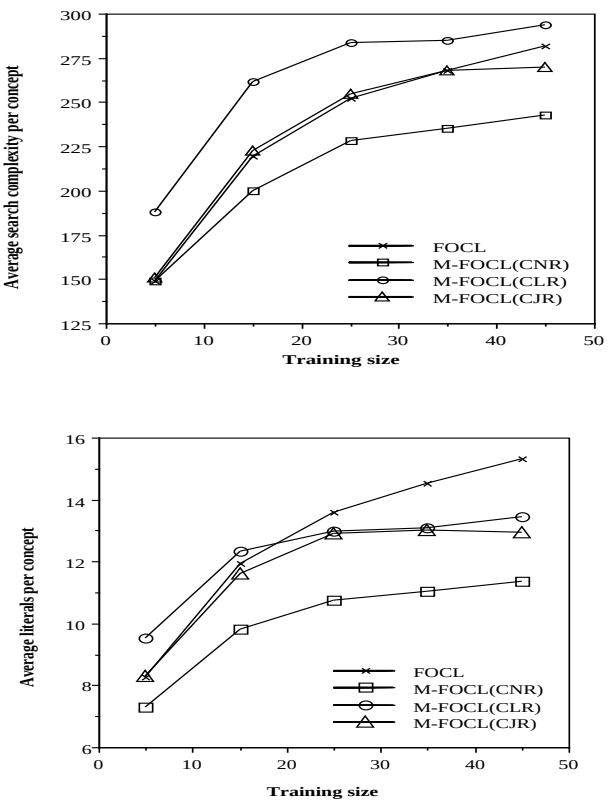


Figure 3: a). Search complexity versus training size. b). concept size versus training size.

Since the M-FOCL methods add background predicates to the description language, the description length of the concepts vary (Figure 3b.). The CNR method consistently used 12% to 25% fewer literals than FOCL to describe the concepts. When compared to FOCL, the CLR method uses 15% more literals for training size 5, and 3% more for training size 15, but uses 5% to 12% fewer for the remaining training sizes. Lastly, the CJR method uses about the same number of literals for training sizes 5 and 15, but then uses 5% to 15% fewer for the remaining training sizes. Thus for larger training sizes, the M-FOCL methods' concept description size decreases with relation to FOCL.

4.3.2 Learning chess moves with tuple noise

To test the robustness of our premise, we also experimented in the chess domain with attribute noise. We measure the amount of noise for a relation in the following manner. Associated with each relation is the set of tuples that satisfy the relation. Altering an element of the tuple is referred to as tuple or attribute noise. When creating attribute noise in the data, for each element in a tuple there is an $n\%$ chance that the element's value is assigned randomly, where n is the amount of attribute noise in the data (Quinlan, 1986).

In the noise test, we experimented in the chess domain with a training size of 25 examples and varied the amount of noise. We used noise levels of 10%, 20%, 30%, and 40%, and used a testing size of 200 examples. We ran 30 trials of each noise level and used the same ordering for the concepts as the previous chess experiments. Note that if an example attribute is noisy due to attribute noise, this does not necessarily misclassify the example.

The CLR and CJR methods had higher accuracy than FOCL for all training sizes[2]. Figure 4a. shows the accuracy for methods CLR, CJR, and FOCL. Although these methods have a higher accuracy, they also search a larger space. Method CLR searches a space approximately double the size of FOCL, while method CJR searches a space from 6% to 31% larger than FOCL's space. The size of CLR's description length was similar to that of FOCL's; however, CJR used from 11% to 13% fewer literals than FOCL to describe concepts (Figure 4b).

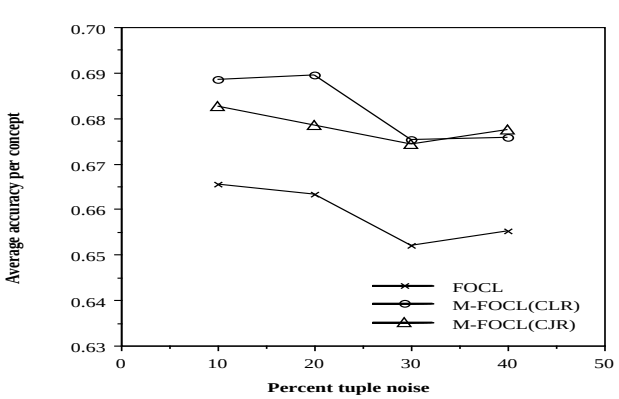


Figure 4: a.) Average accuracy vs. attribute noise.

[2]Method CLR is statistically significant from FOCL with a 97.5% confidence level for all training sizes. Method CJR is statistically significant from FOCL with a confidence level of 90% for 10% and 20% noise and 97.5% for 30% and 40% noise. Method CNR is not statistically significant than FOCL.

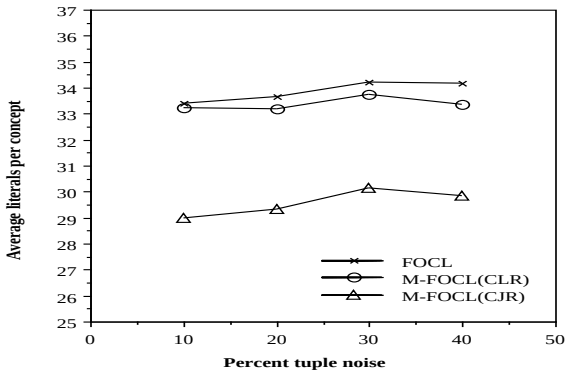
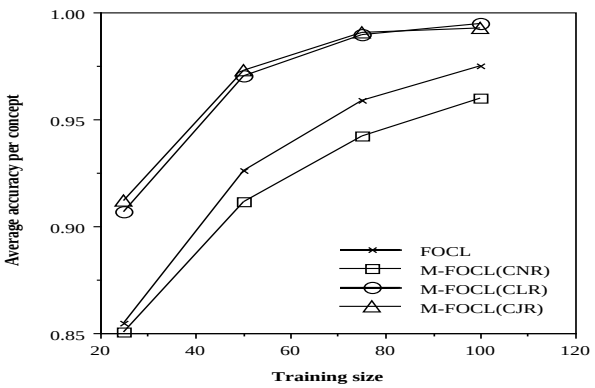


Figure 4: b.) Average literals vs. attribute noise.



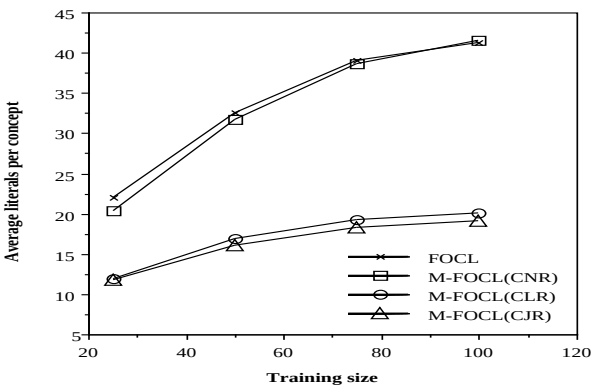


Figure 5: a.) Average accuracy vs. training size. b.) Average literals vs. training size.

4.3.3 Learning poker concepts

We also tested FOCL against M-FOCL in the poker domain. We ran 31 trials of training sizes of 25, 50, 75, and 100 each. We applied the same method used in the chess domain to create the training sizes, that is, we first created a training size of 25 and ran the systems on it, and then we added 25 more examples to create the 50 size training set. All training sets were tested using 500 positive and negative examples each. We used the following concept ordering for the three M-FOCL methods: pair, three-of-a-kind, four-of-a-kind, two-pair, and full-house.

Since the two domains tested have different characteristics, the results of the M-FOCL methods differ from the chess domain. The CLR and CJR methods were up to 5.5% more accurate than FOCL while the CNR method had a slightly lower accuracy than FOCL [3]. Figure 5a. shows the difference in accuracy for the M-FOCL methods. FOCL searched a smaller space than the M-FOCL methods. The CNR method searched a similar sized space to FOCL, while the other two methods searched a space 5 to 10 times greater than FOCL's search space. The CLR method searched the largest space because for each of the concepts, several clauses were being learned and made into clause rules, increasing the branching factor of the search. The learned concept size also varied for each of the systems. The CNR method used approximately the same number of literals to describe concepts as FOCL, while the other two methods used up to 48% fewer literals than FOCL (Figure 5b.). Methods CLR and CJR perform similarly judging from their accuracy and concept description size. In this domain the reoccurring conjuncts closely resemble the clauses learned, thus the subconcepts created by the two methods are very similar, explaining their similar performances.

Judging from the accuracy, search complexity, and concept size, the CNR method performs slightly worse than FOCL. This results mostly from the representation we used for this domain. The representation makes the card ordering important when in reality it is useless information. This information, however, was used when the concept rules were defined, making learning with the concept rules difficult. Although the commutative constraint was used to reduce the search space for the CNR method, it could not compensate for the irrelevant orderings of the cards in the concept rules. This ordering information also affected the other methods. The commutative constraint was not used for the clause and conjunct rules since the definition of any concept sharing rules were not known beforehand and therefore, we did not know if the commutative properties would be applicable. If the commutative constraint was available for the CLR and CJR methods, their search space would have been greatly reduced.

[3] All of the M-FOCL methods are statistically different from FOCL with 99% confidence.

5 Related Work

Currently the idea of learning multiple concepts during classification, and biasing reuse of conceptual structures is a fairly unexplored area. Although there is not much work related to multiple concept learning in classification systems, we will describe related work in other areas of AI.

Systems using analogy (Falkenhainer, Forbus, & Gentner, 1986;De Raedt & Bruynooghe, 1992) and case-based reasoning systems (Branting, 1991; Kolodner, 1989) use additional sources of information either from another domain or from past cases. Since they are using different sources, the information must be generalized in some manner. Systems using analogy use a mapping from the source domain to the target domain. This global mapping defines the transformations from the old domain to the new domain. The mapping may generalize variables and predicates, keeping only the structure of the source domain. Cased-based reasoning retrieves cases from its past cases and generalizes by applying adaptation operators to locally modify the case. Thus for each new case, an old case must be retrieved and a local modification to the old case must be performed. Unlike these two methods of reasoning, concept sharing does not require generalizations for using past concepts to aid learning subsequent concepts. In addition, unlike case-based reasoning methods that require entire cases to be used for matching rather than partial cases, concept sharing emphases sharing any amount of information among concepts.

Learning multiple concepts allows the system to refine its representation language dynamically. Other systems that revise their language apply a variety of methods to create new predicates. STAGGER (Schlimmer, 1987) heuristically searches through the space of conjunctions, disjunctions, and negations of the primitive predicates, using a failure driven test and the examples to see if the new constructive predicates should be added. FRINGE (Pagallo & Haussler, 1990) revises its language by finding Boolean conjuncts of features at the fringe of its decision tree and creates new features from these conjuncts. By modifying the language dynamically, it avoids the replication problem (Pagallo & Haussler, 1990). Although M-FOCL has this self-transference ability, it has a broader scope since it also finds repetitions using other concepts in the domain rather than just within one concept. In addition, M-FOCL differs from these systems since it uses a more relaxed method of modifying the language, that is, it waits for regularities to occur before adding new descriptors to the language.

The theory revision system, EITHER (Mooney & Ourston, 1991) creates intermediate concepts using two operators, inter-construction and absorption, that are similar to the conjunct and concept rules used in M-FOCL. Although both systems use similar operators, they behave differently. EITHER uses a propositional language to describe its categories and ID3 as its inductive component, while M-FOCL uses a relational language and FOCL for its inductive learning. More importantly, M-FOCL does not use any theory initially, but, EITHER begins with an incomplete background theory. In addition, M-FOCL applies its rule operators during concept learning instead of afterwards as in EITHER. Lastly, EITHER performs a post analysis on the rules created to simplify its theory.

These and other systems dealing with predicate formation often generate predicates judged by their current usefulness, while M-FOCL uses past and current usefulness as a generation method. In addition M-FOCL uses information gain as an acceptance criteria for using new predicates in the learned concept description. Thus M-FOCL uses a two level filtering method for applying new predicates to learning.

6 Discussion

We have presented a broader perspective on classification that advocates learning concepts in the context of other concepts in the domain. We implemented this idea by transferring information from learning previous concepts to learning subsequent concepts in M-FOCL. In particular, by using the three rules for transferring knowledge from previously learned concepts, preliminary results show that M-FOCL can learn concepts more accurately, and under some circumstances, more efficiently than without concept sharing. Using M-FOCL to learn in the chess domain and the poker domain provides an illustration of the learning method and evidence for our premise that concepts in the same domain share information.

Concept sharing does have its limitations. If concepts do not share any subconcepts concept sharing will, in the best case, decrease efficiency, and in the worst case, could mislead the system into learning a less accurate description. For example, in the chess domain, the knight legal move did not use many subconcepts from the other concepts, and its subconcepts could not be used by other concepts. Learning the knight move in the context of the other chess concepts increased the search needed to find a concept since useless predicates were added to the language. Although the knight move did not share subconcepts with the other concepts, its presence did not decrease the accuracy of the learned concepts.

This paper concentrates on the sharing of substructures among concepts described relationally. We also intend to explore the usefulness of sharing when concepts are represented using an attributional or a neural network representation. We anticipate similar results; namely by enlarging the search space to include pieces of previously learned concepts, we expect that learning time will sometimes improve and other times

degrade. However, we strongly expect that accuracy will always increase and concept description size will often decrease, as happened with these relational domains. In addition, we hope to characterize the usefulness of concept sharing to better predict situations where this technique would be beneficial for the learning process. The long term goal of this research is to merge concept learning and problem solving to enable systems to learn a domain. Learning a domain may include, among other aspects, learning multiple concepts, learning relationships among the concepts, and learning ways of problem solving using these concepts. We feel that concept learning is a method for defining a problem solving situation and naming the entities in the problem. By adding problem solving techniques, such as planning, we envision a more integrated system that has greater capabilities than either technique alone.

Acknowledgments

We would like to thank Kamal Ali, Cliff Brunk, and Mike Pazzani for their help with FOCL.

References

Branting, K. L. (1991). Exploiting the Complementarity of rules and precedents with reciprocity and fairness. In *Proceedings: Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann.

De Raedt, L., Bruynooghe, M. (1992). Interactive concept-learning and constructive induction by analogy. *Machine Learning*, volume 8, no. 2. Boston, MA: Kluwer Academic Publishers.

Falkenhainer, B., Forbus, K., Gentner D. (1986). *The structure-mapping engine.* (Technical Report no. UIUCDCS-R-86-1275). Urbana, Illinois: University of Illinois at Urbana-Champaign, Department of Computer Science.

Kolodner, J. L. (1989). Judging which is the "best" case for a case-based reasoner. *Proceedings: Case-Based Reasoning Workshop.* San Mateo, CA: Morgan Kaufmann.

Markman, E. (1989). *Categorization and naming in children.* Cambridge, Massachusetts: The MIT Press.

Minton, S., Carbonell, J. G. (1987). Strategies for learning search control rules: an explanation-based approach. In J. McDermott, ed.,*Proceedings of the Tenth International Joint Conference on Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann.

Mooney, R. J., Ourston, D. (1991). Constructive induction in theory refinement. In Birnbaum, Collins, eds., *Proceedings of the Eighth International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Pagallo, G., Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, volume 5, no. 1. Boston, MA: Kluwer Academic Publishers.

Pazzani, M. J., Kibler, D. (1992). *The utility of knowledge in inductive learning. Machine Learning*, volume 9, no. 1. Boston, MA: Kluwer Academic Publishers.

Quinlan, J. R. (1986). The effect of noise on concept learning. *Machine Learning: An Artificial Intelligence Approach*, volume 2. Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, volume 5 no. 3. Boston, MA: Kluwer Academic Publishers.

Ruby, D., Kibler, D. (1991). SteppingStone: An empirical and analytical evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press/The Mit Press.

Schlimmer, J. C. (1987). "Incremental adjustment of representations." In P. Langley, ed., *Proceedings of the Fourth International Workshop on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Vere, S. A. (1977). Induction of relational productions in the presence of background information. In *Fifth International Joint Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.