Case–based Meta Learning: Sustained Learning supported by a Dynamically Biased Version Space

Jacky Baltes and Bruce MacDonald
Computer Science Department
The University of Calgary
2500 University Drive NW
Calgary, Alberta T2N 1N4, Canada
{baltes,bruce}@cpsc.ucalgary.ca

Abstract

It is well–recognized that in practical inductive learning systems the search for a concept must be heavily biased. In addition the bias must be dynamic, adapting to the current learning problem. Another important requirement is sustained learning, allowing transfer from known tasks to new ones. Previous work on dynamic bias has not explicitly addressed learning transfer, while previous case–based learning research suffers from a variety of problems. This paper presents a method of Case–Based Meta Learning (CBML), in which the cases are concepts, rather than instances, and retrieved similar concepts are used as a skeletal version space to speed up learning. CBML is independent of the concept representation language. The CBML–Clerk system, which learns repetitive operating system tasks, is presented as a demonstration.

Keywords: Dynamic Bias, Case–based Meta Learning

1 Introduction

Autonomous agents must be able to learn the classifications of objects by induction. For example, an autonomous robot must learn concepts such as "sidewalk" and "cliff," so it can behave safely. An intelligent agent in the operating system domain must be able to learn concepts such as a user–specific class of backup files. This could include file names such as *file*`.BAK`, *file*`.CKP`, *file*`~`, *file*`.~`*n*`~`, and `#`*file*`#`, as well as more specific formats (e.g. *file*`.save_21_June_1991` and archived files). Some researchers even argue that learning by induction is the main attribute of an intelligent process [Gol91].

In general, the learning problem is to search the space of all possible concept descriptions for a concept consistent with known examples. As well as the traditional resources measures of time and space complexity, designers of induction algorithms must deal with "sample complexity." The number of examples required to learn a concept, which is related to the user effort in teaching, must be minimized so that the user is not over–burdened [Mac91].

In practical systems it is important to make learning manageable by dynamically biasing the concept search in response to the particular examples available, and the learner's current environment [Hei89, HM91].

This paper argues that case–based meta learning (CBML) provides a powerful and intuitive framework for the implementation of a dynamic bias, and presents the CBML–Clerk, which learns repetitive operating systems tasks from examples. As well as using examples to bias the formation of a concept description, previously learned *concepts* strongly bias the search. It is this latter bias that can effectively reduce the sample complexity in terms of the questions a user must answer during learning. As the system learns knowledge about concepts and their descriptions (meta knowledge), it avoids some of the problems associated with case–based learning systems.

After a review of learning in version spaces, inductive

bias, and the importance of dynamic bias, the paper introduces CBML, and presents a case study with experimental results that show a decrease in the number of questions required.

1.1 Learning in Version Spaces

A *version space* is the set of all concepts consistent with the given examples and counterexamples [GN87]. If there is a partial ordering on the set of concepts by generality (forming a *version graph*) then the space can be represented efficiently by its upper or general (G) and lower or specific (S) boundary. The Candidate Elimination Algorithm [Mit77] is a bi–directional search through a version graph. A positive example forces the S set to be generalized to include it, and removes members of the G set that do not cover it. A negative example forces the G set to be specialized to exclude it, and removes members of the S set that do cover it. The two sets may converge to one "correct" concept. If the boundaries overlap, the version space collapses. This is the result of incorrect example classification or a representation language that is not powerful enough to describe the appropriate concept.

2 Inductive Bias

Both empirical observation and the complexity of inductive learning show that the success of a learning algorithm depends on the method used to restrict the hypothesis space (i.e. the inductive bias) [Mit80, Mic83, Hau86, HM91]. Version spaces are an appropriate framework in which to examine bias, particularly dynamic biases, since the candidate elimination algorithm maintains *all* consistent hypotheses, and the result is independent of the order in which examples are presented. Once the version graph is defined, there is no further bias imposed (although the two boundaries move closer as more examples are seen).

Informally, an inductive bias can be defined as any method that prefers one hypothesis over others. Three different categories can be distinguished, based on the implementation mechanism.

- An *absolute* bias is a restriction in the representation language. Certain concepts are not expressible, as they are not in the hypothesis space. A popular absolute bias is the restriction to conjunctive concepts; conjunctions of feature–value pairs.
- A *relative* bias selects one hypothesis over another if both of them are consistent with the example set.
- A *random* bias would select a consistent hypothesis at random.

This paper concentrates on relative biases since they do not prevent the system from learning any concept. Commonly used relative biases are based on the complexity of the concept description or the number of relevant features. For example, given 1, 3, and 8 as positive examples, a learner may prefer the concept "any digit" over "any odd digit or 8." Later if 4 is given as a negative example, the learner must reconsider. However, it is important to note that a fixed relative bias improves the learning only for the initially preferred hypotheses.

Two other dimensions for the comparison of biases are strength and correctness. A *weak* absolute bias does not rule out many hypotheses, whereas a *strong* bias significantly reduces the hypothesis space. A weak relative bias has a minimal effect on the search order, while a strong one reduces the effective search space markedly. A *correct* bias does not rule out the correct concept to be learned, while an *incorrect* one does. Strength is independent of any particular concept, whereas correctness must be defined relative to a concept. A learning system should have a strong bias that meets a certain measure of correctness for the concepts it must learn.

However, a static bias is not generally useful. Heise [Hei89, HM91] has proposed that the dynamic nature of bias is the primary consideration in the design of real world inductive learners, and shown its usefulness in the ETAR robot system. For example, if a system is trying to learn concepts of file names, it is reasonable to prefer prefixes `test*` and suffixes `*.c` over concepts based on occurrences of single characters such as `*e*`. Many users organize their files by a prefix related to the content of the file (`project1`, `test`, ...) and a suffix indicating its type (`.c`, `.tex`, `.txt`, ...). However, if the same system is to learn concepts in other operating system domains, such as string manipulation in a text editor, it may need to learn concepts such as "punctuation symbol," or "any word with the letter `Z` in it" (`*Z*`). Here the prefix/suffix preferences are inappropriate, although the learner should not discard them; it may need to read in files using the file name concepts it knows. Furthermore, it is conceivable that some users organize their files in a different way, for example placing different file types in different directories, so that preferred concepts might include `project/*/test` or `project/*/a.out`. Therefore, the bias must be dynamic so that the learner can adapt to both the user preferences and the current context.

3 Systems that use Dynamic Bias

STABB Utgoff argues that there are three steps in bias adaption [Utg86].

- *Recommend* new concept descriptions that should be added to the hypothesis space.
- *Translate* the recommendations into expressions of the concept description language.
- *Assimilate* newly formed concepts into the original hypothesis space.

STABB represents the concept space as a version graph. An overly strong bias will remove the required concept, and this is detected when the version space collapses (i.e. there are no more hypotheses that are consistent with all examples). STABB then adds extra nodes to the version graph. One heuristic for new node selection is to add the least disjunction required, as depicted in figure 1 where the concept `sin` $\vee$ `cos` is added. Assimilating the new concept is not straightforward for STABB because adding nodes to a version graph may change the boundaries.

Predictor Gordon's system uses a feature value representation for instances [Gor90]. An example object in the domain can be represented as

```
object = (mat=wood,size=large,shape=sphere).
```

The Predictor system uses three assumptions to adapt the bias.

- *Irrelevance* is used to *mask* features from the concept description (e.g. ignore the material of an object).
- *Cohesion* determines when to climb a generalization hierarchy of a feature. For example, in the version space of figure 1, if *sin* is the only example, *cohesion* will try to generate the concept description *trig*.
- *Independence* is used to mask feature–value pairs from the representation language. Two features are independent if and only if you can independently change one of the feature values and the resulting object is also a member of the concept. If `obj1=(red, block, wood)` and `obj2=(green, sphere, wood)` are positive examples, then `color` and `shape` are independent if and only if `obj3=(green, block, wood)` and `obj4=(red, sphere, wood)` are positive examples as well.

Predictor tests whether any of the biasing assumptions can be applied and then actively tries to verify this assumption. Although it is an incremental learning system, it can take advantage of the set of examples supplied. For example, if some objects satisfy the *irrelevance* criteria for a given feature, the Predictor system searches the set of examples for instances that verify or invalidate this assumption. If the assumption is verified for the current example set, the bias is adjusted and the hypothesis space reduced by masking the *irrelevant* feature. If in the future other examples show that the feature is not irrelevant, it will be unmasked.

ETAR Heise's system uses dynamic bias to learn robot procedures from examples [Hei89, HM89]. In contrast to STABB, ETAR strengthens the bias by focusing on aspects of a task trace that meet a relevance criterion. The criterion is spatial locality relative to the robot hand. This enables a raw numeric sequence of teacher guided robot positions to be partitioned into a chain of symbolic action nodes. The bias also enables action nodes to be merged when there are similar nodes within and between example task sequences, thus enabling the determination of branches and loops in the task. Throughout, the bias is driven by the example, or task trace, and dominates the search for an appropriate procedure. Example tasks include blocking stacking, and sorting objects from a conveyor.

4 Issues in Dynamic Biasing

The common approach in the STABB and the Predictor systems is to use heuristics to adapt the representation language and thus the hypothesis space. ETAR on the other hand uses its bias to modify the input example information, and the search process. All three address the problem of learning a single concept, not taking advantage of previous learning in a new learning episode.[1] This has two limitations. In an autonomous agent, rather than defining different representation languages and different biases for each concept, we would like to implement a general purpose learning algorithm that supports transfer of concepts. Second, the choice of initial bias has been ignored. The STABB system starts out with a strong bias and weakens this bias if forced to. Predictor uses a weak bias and tries to strengthen it. This means that both systems require extra work to achieve a successful bias. A system should use previous experience to approximate the initial bias.

The systems' biases operate directly on particular rep-

[1] Although Heise's current work addresses this (personal communication).

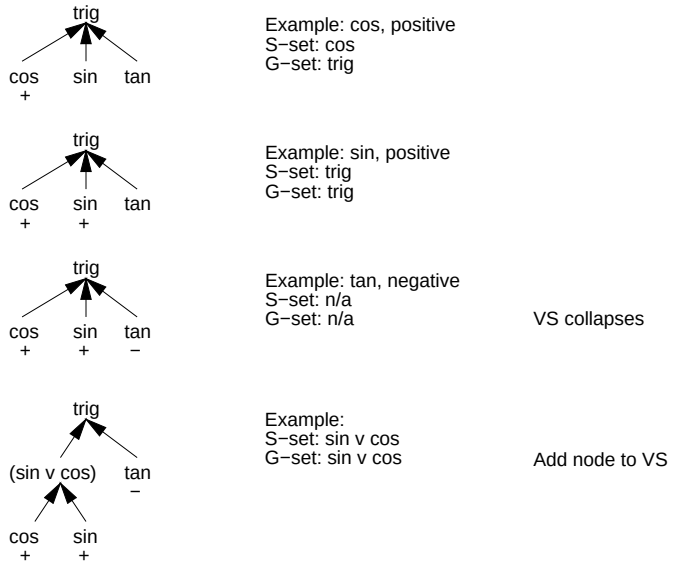

Figure 1: Dynamic Bias in the STABB System. A least disjunction is added when the space collapses.

resentation languages. If the language is changed, for example from DNF to CNF, then the dynamic biasing algorithm must be updated. In addition, since transformations of the representation language are made explicitly in STABB and Predictor, these approaches require explicit adaptation rules. Such rules may be hard to find and to compute for non–trivial representation languages. For example, the Predictor's *independence* biasing assumption is hard to represent in DNF; the concept for independent `color` and `shape` in the description above would be: `(red and block) or (red and sphere) or (green and block) or (green and sphere)`. The representation is simpler in CNF: `(red or green) and (block or sphere)`.

CBML enables task transfer, and is not specific to particular representation languages.

5 Case–based Learning

In case–based learning (CBL) classified examples (cases) are stored and used to help in classifying new, unknown examples [Aha91]. The focus is on different issues than other case–based reasoning methods. The case representation is restricted to feature–value pairs and cases are not adapted to fit a new situation. A simple indexing scheme uses a similarity assessment of the new and previous cases. Approaches range from encoding a large amount of domain knowledge (e.g. Protos [Bar89]) to computing similarities dynamically (e.g. MBRtalk [SW88]).

Breiman *et al.* [BFOS84] argue that simple CBL algorithms are computationally expensive (because similarities between all cases and the current concept must be computed), intolerant to noise and irrelevant features, sensitive to the similarity function, give no simple way to define similarity functions for symbolic–valued features, and provide little information about the structure of the data. Aha [Aha91] proposes methods to overcome the first two, and notes that CBL must be sensitive to the current context. GCM–ISW [AG90] uses context as well as goal features in the similarity assessment of new cases.

A major criticism of CBL is the large amount of storage required. Recent research has alleviated this problem by storing only instances that can discriminate among different classifications [Aha91]. In this paper cases do not represent single instances, but concepts learned in previous tasks. The case memory grows only linearly with the number of tasks that the system learns. The justification of this approach is analogous to Hammond's justification for case–based planning [Ham89]. Learning is an expensive operation so the results of the learning procedure should be stored and reused in the future.

6 Case–based Meta Learning

A robot should be able to use previously learned concepts when learning a similar concept in the future. In CBML previous cases are used as a skeleton of the hypothesis space, to guide the search. The skeletal hypothesis space consists of concepts that have been

successfully used in the past on a similar task. Once the most appropriate known concept is found, domain learning algorithms are used to find the correct node.

CBML operates independently of the learning domain and algorithm, except for the specification of the learning context. Previous cases are retrieved using the learning context, which may be based on the following features:

- **Example set:** Retrieve concepts that were used previously to classify a similar example set.
- **Task description:** If the learner is given a description of the task, retrieve concepts that were used in similar tasks.
- **User advice:** The learning context can be established by the user through special instrutions. For example, the user might tell the system that the new concept is similar to known concepts $c1$ and $c5$.
- **Domain knowledge:** Extra knowledge may allow the system to establish a similarity between learning contexts. For example, a system might know that copying and moving files are similar operations.

CBML implements a bias relating a new concept with previous learning experiences and tries to maintain the partitioning of the current instance set that would be imposed by previously know concepts. The bias is to maintain sets of instances that were learned by previous similar cases. For example, if each previous case generates one consistent classification (positive or negative) for all elements of the current instance set matching `*.txt`, then the preferred hypotheses are those that assign one particular classification to all strings ending in `.txt`.

One distinction between previous dynamic biasing systems and CBML systems is that CBML bias is adjusted only when learning a concept in a similar task. It cannot yield better performance if the system is used only to learn one specific concept. However, CBML and other dynamic biasing algorithms can complement each other since CBML is independent of the specific learner/classifier algorithm. Furthermore,

1. CBML supports context dependent biases. Previous concepts that were useful in executing similar tasks are retrieved. Different tasks can use a common representation language.
2. CBML can provide an initial bias. If all similar previous concepts assign the same classification to all strings matching `*.txt`, then the important feature of this task may be suffixes.
3. CBML does not change the representation language explicitly, is independent of the language and the concept learner/classifier, and needs no knowledge of how to change the bias to focus on, say, suffixes.
4. Since transformations are made only implicitly, transformation rules are not necessary.

CBML also does not suffer from the problems usually associated with CBL. The computational complexity is reduced since only concepts and not instances of concepts are stored. Noise in a CBML system is equivalent to retrieving a case that uses a different bias than the concept to be learned. This affects the efficiency but not the correctness of the learning procedure. Instead of retrieving one similar case, CBML systems retrieve all similar cases. Therefore, a CBML system is more robust with respect to irrelevant features and the choice of the similarity function. A CBML system uses a learner/classifier routine for learning and thus does not need to represent symbolic-valued features or structured data at an instance level.

7 CBML Algorithm

This section describes the CBML algorithm. As an illustration, an example similar to Diana Gordon's [Gor90] object domain is used. The representation language is shown in figure 2. Objects are described as feature/value pairs. The three features of an object are its material, size, and shape. For example `(element,any,brick)` describes the concept of aluminum or copper bricks of any size.

The input to a CBML system is a learning context, a retrieval function, a set of instances that must be classified, a memory of previous concepts, and a learner/classifier algorithm. The learner/classifier routine classifies instances as *positive*, *negative*, or *unknown*. If provided with the correct classification (*positive* or *negative*) of an *unknown* instance, it returns an updated concept that is able to classify the previously *unknown* instance in the future. The set of instances is assumed unclassified and CBML attempts to ask the minimum number of questions to learn the unknown target concept over the instances.

Table 1 describes the CBML algorithm. After retrieving all similar cases from the concept memory (Step 1), the classification of all instances by all similar cases is computed (`for-each` beginning Step 2). If none of

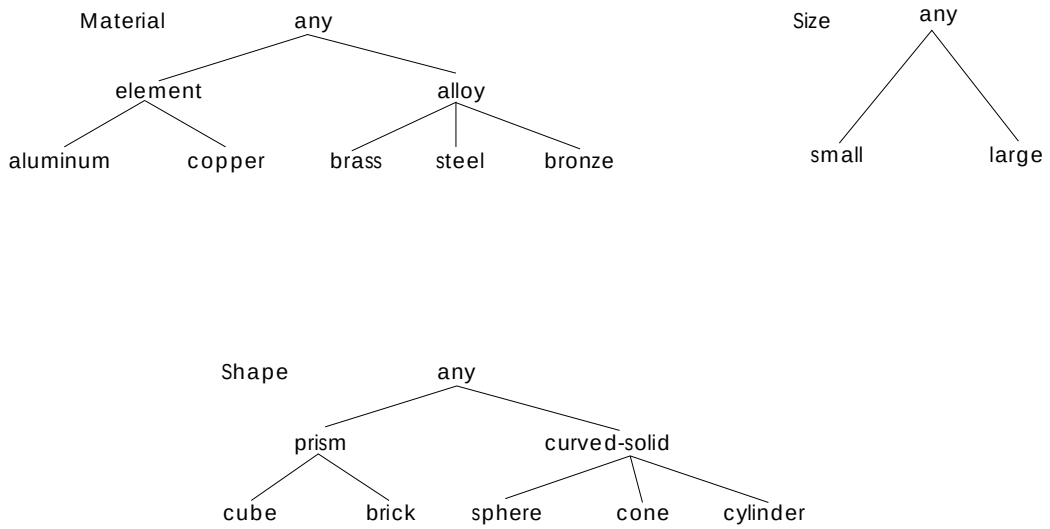

Figure 2: Version Spaces for Object Example

Table 1: The CBML Algorithm

```
CBML(learning-context,find-similar,instance-set,memory,learner/classifier)
        (returns an updated case memory)

1.    similar-cases := find-similar((learning-context), memory)
      new-concept := nil-concept
      unknown-instances := nil

      while (similar-cases not empty)
2.       for-each case in similar-cases
3.          if inconsistent(instance-set, case) then
               remove(case, similar-cases)
            case' := make-case(learning-context,concept(case))
            (_, class) := learner/classifier(case', instance-set)
4.          if all-instances-classified(class) then return(update-memory(case'))
5.          replace(similar-cases,case,update(concept(case),class))
6.       remove-equivalent(similar-cases)
7.       bmm := best-min-max(similar-cases)
8.       if ask-user(instances(bmm)) = TRUE or FALSE
9.          then new-concept := update(new-concept,instances(bmm))
10.         else unknown-instances :=
                    append(unknown-instances,
                           remove(instances(best-min-max), instance-set))

11.   (new-concept,_) := learner/classifier(
                           new-concept,
                           append(unclassifiable-instances(instance-set),
                                  unknown-instances))
      return(update-memory(new-concept))
```

`learner/classifier` returns a new concept and the classification partition made by it. `concept(case)` returns the string concept of the case. A case includes a command, a prototype string and a string concept.

the instances in the set are *unknown*, then there is no need to learn. The algorithm simply returns after adding the concept to the case memory (Step 4). Similar cases are removed if inconsistent with the current instance set (Step 3).

In the machine shop example given in table 2, the system has previously learned the use of three different tools for polishing objects and for what objects those tools are applicable. The new concept, which is still unknown to the system, describes objects that can be polished using a fourth tool `C=(alloy,small,prism)`. In the example, the learning context is determined by the task description (polishing objects). Therefore, previously learned concepts for polishing objects are retrieved.

In the next stage, similar cases are updated with the information from the current instance. For example, if case c is unable to classify instance i, but the current concept classifies i as either *positive* or *negative*, case c is updated to include the classification of i. Note that this update is temporary and is not reflected in the case memory (Step 5). In the example, this step is omitted, because all retrieved concepts are complete, that is they classify all new instances as either positive or negative.

The algorithm next removes decision equivalent concepts (Step 6). Two concepts are decision equivalent if they yield the same classification relative to all instances in an instance set. For example, if the algorithm had retrieved another concept `C4=(copper,any,prism)`, `C3` and `C4` are decision equivalent for the instance set in the example. One of the concepts would be removed.

In the next step, the system is trying to find a subset of the instance set that allows it to discriminate best among all similar cases. Then it can ask the user the most useful question (i.e. the classification of that set). For this purpose, a two–dimensional array is constructed, called the Min–Max–Table (Step 7). The rows of the Min–Max–Table consist of sets of instances that have the same classification under all similar cases and are classified as *unknown* for the new concept. The columns consist of the similar concepts. The entry at row i and column j is the classification of instances in set i under the concept j. Table 2 gives an example of a Min–Max–Table.

If there is only one element in the instance set for a given row, the row is deleted from the Min–Max–Table because it will only require one question to the user (at a later stage) to find the classification of this instance (In the example, `(copper,small,brick)` and `(bronze,big,brick)` are removed).

Selecting an optimal ordering of sets of instances (giving a minimum number of questions) is equivalent to inducing an optimal decision tree. This problem is exponential. The CBML algorithm uses a local decision procedure to find the best set of instances based on their Min–Max–Value.

All instances belonging to the set with the best Min–Max–Value are then presented to the user with the request to classify *all* of the instances as *positive*, *negative*, or *neither*. If the answer is *positive* or *negative*, the concept to be learned is updated with the new classification for all elements of the set. If the user answered *neither*, these instances are removed from the instance set (Step 8). The user will be asked about the correct classification for all these instances later (Step 11).

The heuristic used to select the best Min–Max–Value is based on the assumption that the response of the user will often be *positive* or *negative*, so that a large number of similar cases will be removed. Let us assume that a set of instances is classified as *positive* by two similar cases, as *negative* by three cases, and as *unknown* by one case. Let us further assume that the size of the set of instances associated with this row in the Min–Max–Table is three. If the user's classification is *positive*, three cases can be removed from the Min–Max-Table, because they are inconsistent with this information. On the other hand, if the classification is *negative* only two cases can be removed. If the user answers *neither*, no concept can be removed and the user must be asked for the classification of these instances separately.

Min–Values and Max-Values are calculated for each row. The CBML algorithm selects the next instance set according to these rules:

- Choose the maximum of the Min–Values of all rows. The Min–Value of a row is the minimum of the number of *positive* classifications and *negative* classifications in a row.
- If there is a Min–Value tie among rows, select the instance set (row) with the highest Max–Value. The Max–Value is the maximum of *positive* and *negative* classifications.
- If there is still a tie among rows, select the larger instance set.

In the example, the system will select the second row of the Min–Max–Table. The system will ask the user whether `(brass,small,brick)`,

Table 2: Example 1: Machine shop domain. Shown below are concepts, instances, Min–Max–Table, and a sample dialog for learning a new polishing task

Retrieved concepts	`C1=(alloy,any,curved)`
	`C2=(element,any,prism)`
	`C3=(any,small,brick)`

New Concept	`C=(alloy,small,prism)`

New Instances	`(brass,big,sphere)` `(steel,small,cylinder)`
	`(brass,small,brick)` `(steel,small,brick)`
	`(bronze,small,cube)` `(copper,small,brick)`
	`(bronze,big,brick)`

Min–Max–Table

Instance	C	C1	C2	C3	Min–Max
(brass,big,sphere)	?	t	f	f	1,2
(steel,small,cylinder)	?	t	f	f	
(brass,small,brick)	?	f	f	t	1,2
(steel,small,brick)	?	f	f	t	
(bronze,small,cube)	?	f	f	t	
(copper,small,brick)	?	f	t	t	removed
(bronze,big,brick)	?	f	f	f	removed

Sample Dialog

```
Are all of the following instances positive, negative or neither?
(brass,small,brick) (steel,small,brick) (bronze,small,cube)
USER> positive
```

`(steel,small,brick)`, and `(bronze,small,cube)` are all *positive* or *negative* examples. The instances in the set can be classified simultaneously.

In the example, the instances in the selected instance set are all classified as *positive*. If the standard candidate elimination algorithm is used, after these three examples, the S–set will have been generalized to `(alloy,small,prism)`. Therefore, it follows that `C1` and `C2` are incompatible with the new concept. Hence, the system tries to apply adaptation rules to concepts `C1` and `C2`, trying to adapt them to the new concept. In this example, assume that no adaptation rules are given to the system. Therefore, the system will remove `C1` and `C2` from consideration. Since not all instances are classified, the system returns to step 2. The resulting Min–Max–Table is shown in table 3. Since `(copper,small,brick)` and `(bronze,big,brick)` again yield a single instance set, their classification is delayed until later. Only a single instance set is remaining. The classification of the second instance set as negative examples, yields a G–set of `(any,any,prism)`.

The user then classifies the only remaining instances `(copper,small,brick)` and `(bronze,big,brick)` as negative. This specializes the G–set to `(alloy,small,prism)` which is identical to the S–set. This means that the correct concept is learned.

8 Implementation of the CBML–Clerk

To show the operation of CBML in a more realistic learning setting, Baltes implemented a CBML system on top of the Shell–Clerk [Bal91], which is an instructable system (see [Mac91]) that learns repetitive operating system tasks by example, such as copying files, arranging mail messages, or reading news articles. File names are represented as strings and the representation language is a subset of regular expressions, limited to at most three terms in a disjunction. Nevertheless, it can learn concepts such as all backup files (`*~ or #*# or *.CKP`), all C source files (`*.c or *.h`), and all test files (`*test* or *Test* or *TEST*`). The system uses a "symmetric version space" (SVS) approach to learning string concepts, but requires a few too many questions (19 for 96 files) to learn common concepts such as `*.*` or `*~`. Based on the argument given in the previous section, CBML–Clerk was designed as an extension.

Instead of using a dynamic bias to select concept descriptions and use these concepts, constraints imposed by the domain require that the learning algorithm must not over–generalize (mistaken file erasure is unacceptable). The CBML–Clerk uses the instance set partitions generated by previous tasks as a way to reduce the number of questions to the user. It will ask for the classification of one of these sets, rather than that of a single example. Although the there is no decrease in the number of instance classifications required from the user, fewer questions are asked.

The Clerk requires the user to begin learning by listing a set of files that contains the target ones, then inputing a command and a prototype file name on which this command is to be executed. The original version then proceeded to question the user about individual file names. The CBML–Clerk begins the same way, then proceeds to ask questions about possibly relevant sets of file names.

8.1 Case Representation

A case records the prototype file name, the command, and the concept description produced by the SVS algorithm. The candidate elimination algorithm can not be used directly because the G set is possibly infinite for a representation language with limited disjunctions in an infinite domain.[2] Concepts are described by a positive *cover set* — the most specific description of all positive instances — and a negative *cover set* — the most specific description of all negative instances — plus other information. For example, a concept matching strings `a`, `b`, and `c` but not any digits or other lower case characters, is represented by the following structure[3]

```
Pos-Cover: (a or b or c)
Neg-Cover: (<digit> or <lowercase>)
```

8.2 Case Indexing

Recalled concepts define the skeletal hypothesis space. A case is considered similar to the current learning problem if either the prototype file name or the command are the same.

8.3 Case Adaption

CBML provides the opportunity to adapt old cases to new situations, because cases are concept descriptions that may be used to partition the instance set. For example, if the concept to be learned (say `*.txt`, although the learner does not know this yet) and a retrieved concept (say `*.tex`) are determined to be inconsistent, an adaption rule can be fired to change

[2] See [Bal91] for details of the SVS algorithm.

[3] The SVS algorithm requires two extra *cover sets*. These can be safely ignored for the discussion.

Table 3: Example 1: After classification of the first instance set.

Min–Max–Table 2

Instance	C	C1	C2	C3	Min–Max
(brass,small,brick)	t	—	—	t	—
(steel,small,brick)	t	—	—	t	
(bronze,small,cube)	t	—	—	t	
(brass,big,sphere)	?	—	—	f	1,2
(steel,small,cylinder)	?	—	—	f	
(copper,small,brick)	?	—	—	t	removed
(bronze,big,brick)	?	—	—	f	removed

the recalled concept description. In CBML–Clerk, if a similar case is inconsistent then the learner examines the complement of the concept, and if this is also inconsistent, then that retrieved concept is removed from consideration.

9 Results

The CBML–Clerk was tested on a sequence of different tasks in the operating system domain. Common concepts such as `*.*, *.c, *.tex, *~` are learned after only a few example tasks. The concept in table 4 is learned after only three questions. Table 5 shows the results for the first five concepts in the original SVS paper [Bal91].

Here the concepts are not learned in any particular order and are not that well related, so that CBML is not particularly effective. Table 6 shows a more significant improvement when concepts are related, and are taught in a reasonable order. In both tests the command was the same for all concepts. Further, more exhaustive tests are underway.

10 Conclusion

This paper argues that skeletal, case–based hypothesis spaces can be used to reduce the sample complexity of learning algorithms. CBML is an simple, efficient, and intuitive way to construct hypothesis spaces from previous experience. By using a case–based approach, the constructed spaces are context dependent and can therefore be used as a dynamic bias.

CBML overcomes difficulties associated with dynamic biasing and case–based learning. An advantage is its robustness, which is gained by retrieving *all* similar cases. Furthermore, CBML does not require an explicit set of transformation rules for adapting bias. CBML is independent of the concept representation language. Although we have yet to devise sophisticated case adaption rules, CBML enables learning without case adaption. Improvements could also be made to the simple indexing scheme. Furthermore, at the moment only a single skeletal version space is generated. It seems reasonable that a hierarchy of spaces could further reduce the search through the hypothesis space, just as abstraction hierarchies do in planning. The algorithm could also be extended to combine similar cases in the case memory, if many are found. This would reduce the computational complexity as well as the storage requirements.

CBML promises to be an appropriate trade-off between the detail of case–based learning, and the generality of inductive inference.

References

[AG90] D. W. Aha and R. L. Goldstone. Learning attribute relevance in context in instance–based learning algorithms. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pages 141–148, Cambridge MA, 1990. Lawrence Birnbaum.

[Aha91] David W. Aha. Case–based learning algorithms. In *Proceedings: Case–Based Reasoning Workshop*, pages 147–158, 1991.

[Bal91] Jacky Baltes. A symmetric version space algorithm for learning disjunctive string concepts. In *Proc. Fourth UNB Artificial Intelligence Symposium*, pages 55–65, Fredericton, New Brunswick, September 20–1 1991.

[Bar89] Ray Bareiss. *Exemplar–Based Knowledge Acquisition.* Academic Press, Inc., 1989.

[BFOS84] L. Breiman, J.H Friedman, R. A. Olshen, and C. J. Stone. Classification and regres-

Table 4: Example 2: CBML–Clerk Min–Max–Table.

Learned concepts	`*.* *.c (*~ or *.bak) (*.c or *.cc)`
New Concept	`*.c or *.h`
New Instances	`test.c test.c~ test.bak test1.c`
	`test.h test.o test.dat a.out`

	new	*.*	*.c	*~ or *.bak	Min–Max
test.c	?	t	t	f	1,2
test1.c	?	t	t	f	
test.c~	?	t	f	t	1,2
test.bak	?	t	f	t	
test.h	?	t	f	?	rem.
test.o	?	t	f	f	1,2
test.dat	?	t	f	f	
a.out	?	t	f	f	

Table 5: Results for the first five concepts reported in the original SVS algorithm.

Concept	Prototype	CBML Q	SVS Q	Comments
test* or #test*	test.ss	32	32	identical to SVS
*	generic.c	2	19	optimal case for CBML
*(file-io or built-in)	built-in_amiga.ss	38	38	does not match previous
(_amiga or _sun)*				prototype; not adapted
(amiga or sun)	built-in_amiga.ss	34	42	
e	test.ss	39	41	

[Bal91]. There are 96 files and the third and fourth columns give the number of questions asked by CBML and the original SVS algorithm.

Table 6: Results for some related concepts, again comparing CBML and the original SVS algorithm.

Concept	Prototype	CBML Q	SVS Q	Comments
*.c	test.c	8	8	For perfect learning it must ask about all
.	test.c	2	8	an optimal case
a.out	a.out	2	8	
*~ or *.bak	test.bak	6	8	
*.c or *.h	test.c	6	8	

There were eight files:
`a.out test.c test.h test.c test.bak test1.c test.o test.dat`.

sion trees. Technical report, Wadsworth International Group, Belmont, CA, 1984.

[GN87] M. R. Genesereth and N. J. Nilsson. *Logical foundations of artificial intelligence.* Morgan Kauffman, 1987.

[Gol91] Lev Goldfarb. Verifiable characterization of an intelligent process. In *Fourth UNB Artificial Intelligence Symposium*, pages 67–80, 1991.

[Gor90] Diana Faye Gordon. *Active Bias Adjustement for Incremental, Supervised Concept Learning.* PhD thesis, University of Maryland, 1990.

[Ham89] Kristian J. Hammond. *Case Based Planning.* Academic Press Inc., 1989.

[Hau86] David Haussler. Quantifying inductive bias in concept learning. Research report UCSC-CRL-86-25, University of California, Santa Cruz, CA 95064, November 1986.

[Hei89] Rosanna Heise. Demonstration instead of programming: focussing attention in robot task acquisition. Master's thesis, Department of Computer Science, University of Calgary, 1989.

[HM89] Rosanna Heise and Bruce A. MacDonald. Robot program construction from examples. In *Proc. National Irish AI Conf.*, Dublin, Ireland, September 1989. Also in book form, edited by A. F. Smeaton and G. McDermott (Eds.), AI and Cognitive Sciences '89, Springer–Verlag, 1990.

[HM91] Rosanna Heise and Bruce A. MacDonald. Dynamic bias is necessary in real world learners. 1991. Submitted to *Journal of Experimental and Theoretical AI.*

[Mac91] Bruce A. MacDonald. Instructable systems. *Knowledge Acquisition*, December 1991. (to appear) Accepted by *Knowledge Acquisition* without revision. A much shorter version was presented at the 1990 Banff Knowledge Acquisition Workshop; see conference papers.

[Mic83] R. S. Michalski. A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonnel, and T. M. Mitchell, editors, *Machine Learning, Vol. 1.* Tioga, Palo Alto CA., 1983.

[Mit77] Tom M. Mitchell. Version space: A candidate elimination algorithm approach to rule learning. In *Proceedings of the 5th International Conference of on Artificial Intelligence*, pages 305–310, 1977.

[Mit80] Tom Mitchell. The need for biases in learning generalizations. Technical Report TR CBM-TR-117, Rutgers University, 1980.

[SW88] C. Stanfill and D. Waltz. Learning to read; a memory–based model. In *Proceedings of a Case–Based Reasoning Workshop*, pages 402–413, 1988.

[Utg86] Paul E. Utgoff. *Machine Learning of Inductive Bias.* Kluwer Academic Publishers, 1986.