Factorization in Experiment Generation

Devika Subramanian
Joan Feigenbaum

Department of Computer Science
Stanford University
Stanford, CA 94305

ABSTRACT

Experiment generation is an important part of incremental concept learning. One basic function of experimentation is to gather data to refine the existing space of hypotheses[DB83]. Here we examine the class of experiments that accomplish this, called *discrimination experiments*, and propose *factoring* as a technique for generating them efficiently.

I Introduction

The need to generate experiments that discriminate between sets of hypotheses arises in the context of a learner using the version space algorithm[Mit78][Mit83]. Here we show how implicit independence relations in the concept language can be used to factor the version space of hypotheses. We analyze the computational advantages gained by doing experiment generation using the factors.

This paper is organized as follows. Section II describes the single concept learning problem that provides the setting for our investigation of the discrimination-experiment-generation(DEG) problem. Next, we introduce the blocks world example that will be used throughout this paper. In Section IV we characterize the DEG problem and explain why it is hard. Section V briefly describes two sources of information that can be used to make it tractable; one is domain-specific information, the other is knowledge of independence between parts of the concept being learned, which is domain-independent. In this section we also outline how this independence allows us to factor the version space and generate experiments by working with the factors. The next two sections are a formal analysis of factoring: Section VI demonstrates the conditions under which a version space can be factored (under the independent credit assignment (ICA) assumption) and provides an optimal strategy for generating experiments in the factored space. Section VII does a similar analysis for the case that ICA is not available. The tradeoffs associated with factoring along with a cost comparison are presented in Section VIII. Experimental results using our implementation of factoring are sketched in Section IX. Finally, Section X highlights the main contributions of this paper and concludes with a proposal for future work on this problem.

II The Single Concept Learning Problem

The *single concept learning problem*[Mit78] is:
Given:

- a first order *concept language* C,
- a first order *instance language* I,
- a set P of sentences in I, containing positive instances of the concept to be learned,
- a set N of sentences in I, containing negative instances of the concept to be learned,
- the $\models$ relation between sentences in C and I that indicates when an instance matches a concept (i $\models$ c).
- a biasing theory T that describes which of several alternative descriptions of a concept is more plausible.

Find: the concept description (represented as a sentence c in C) that is consistent with (P,N), i.e.

- $\forall$ p. p $\in$ P $\Rightarrow$ p $\models$ c
- $\forall$ n. n $\in$ N $\Rightarrow$ n $\not\models$ c

Typically, the learner is given sets P and N that fail to determine c uniquely(that such a c exists follows from the *representability assumption* made in the version space algorithm [Mit78]), thus the learner constructs the set VS of descriptions that are consistent with the observed instances. VS is called the *version space* of the concept to be learned. The learner now attempts to gather more information about the concept by using instances in I (that will be classified by a critic/teacher) to eliminate some of the concept descriptions in VS. This process is iterated until a single concept description survives. Finding the sequence of instances in I that will accomplish this is the *discrimination-experiment-generation problem*. If more than one concept description remains and the instances in I do not tell them apart, the learner uses T to select the most plausible one.

III Blocks World Example

The single concept learning problem will be illustrated with the following blocks world example. We define the vocabulary of the concept language C_1. The constants of C_1 are

- `X, Y, Z, ...` (names of blocks)
- `red, green, any-colour` (colours of blocks)
- `cube, brick, wedge, pyramid, any-shape` (shapes of blocks)

The predicates of C_1 are

- `shape`: name of block x shape of block $\rightarrow$ {T,F}
- `colour`: name of block x colour of block $\rightarrow$ {T,F}

The pure predicate language constructed from this vocabulary is C_1. The following are relations between well-formed formulae in C_1.

```
(if (shape $x brick) (shape $x wedge))
```

The first author is supported by an IBM fellowship. The second author's work was supported by a Xerox fellowship.

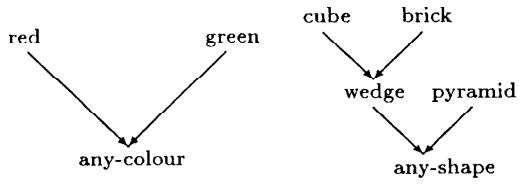

Figure 1

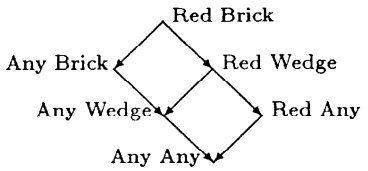

Figure 2

```
(if (shape $x cube) (shape $x wedge))
(if (shape $x wedge) (shape $x any-shape))
(if (shape $x pyramid) (shape $x any-shape))
(if (colour $x red) (colour $x any-colour))
(if (colour $x green) (colour $x any-colour))
```

They are used to construct generalizations/specializations of a concept in C_1. If x logically implies y then x is said to be *more-specific* than y. Logical implication defines a partial order on the sentences of C_1. A concept is generalized by constructing its logical consequences. A concept is specialized by constructing the sentences in C_1 that it can be deduced from.

We can express these relations with the *directed acyclic graphs* (DAGs) in Figure 1.

We use the shorthand notation `cube` for `(shape $x cube)` ... etc. and `red` for `(colour $x red)` ... etc.

Without loss of generality, we assume $I \subset C$. This is referred to as the *single representation trick*. However, we do not regard it as a trick, because the only other alternative under the representability assumption is for I to be the domain of an injective mapping whose range is a subset of C. I can be regarded as the observational component of C.

In our example,
I = {`red cube, red brick, red pyramid, green cube, green brick, green pyramid`}

Suppose the learner is presented with C and I as above as well as initial values:

- P = { `red brick` }
- N = $\emptyset$
- T = $\emptyset$

The learner constructs the VS in Figure 2 using the relations in C_1. Each node of the graph is a sentence in C_1. Each arc stands for logical deducibility.

We describe how the version space is updated in response to a labelled instance. Suppose the learner asks the teacher whether `red cube` is a positive instance. The two possible updates to the VS are shown in Figure 3.

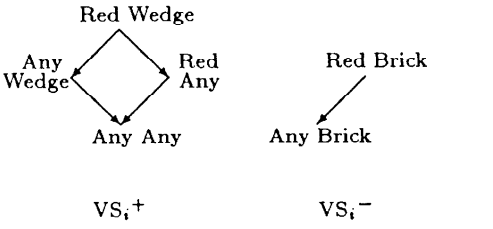

Figure 3

Formally,

- $VS_i^+ = \{ c \mid c \in VS \text{ and } i \models c \}$
- $VS_i^- = VS - VS_i^+$

This is called candidate elimination[Mit78].

IV The Discrimination Experiment Generation (DEG) Problem

The DEG problem is that of finding a minimal sequence of instances in I that will cause the VS to converge to a single concept description. We study this problem under the following assumption.

- All hypotheses in the VS are equally likely. This means that the learner has no *a priori* basis for preferring one hypothesis over another (i.e. T = $\emptyset$).

A *strategy* for DEG is a policy for choosing the next instance in the experiment sequence. [Sub84] gives a formal proof that the general DEG problem is NP-hard and that the strategy presented below is optimal under the assumption above.

Given that all hypotheses are equally likely, the probability p_i^+ that an instance i in I is a positive instance of the concept is:

$$p_i^+ = \frac{|VS_i^+|}{|VS|}$$

The probability that i is a negative instance of the concept being learned is:

$$p_i^- = 1 - p_i^+ = 1 - \frac{|VS_i^+|}{|VS|} = \frac{|VS_i^-|}{|VS|}$$

The expected size of the version space if i is chosen as the next instance in the experiment sequence is

$$\begin{aligned} E(i, VS) &= p_i^+ |VS_i^+| + p_i^- |VS_i^-| \\ &= \frac{1}{|VS|}(|VS_i^+|^2 + |VS_i^-|^2) \end{aligned}$$

Notice that an instance that all hypotheses match (resp. don't match) has $p_i^+ = 1$ (resp. $p_i^- = 1$). Such an instance has E(i,VS) = |VS|, confirming our intuition that it has zero discriminatory power. The function E(i,VS) has a minimum value of $\frac{|VS|}{2}$ which is achieved when $|VS_i^+| = |VS_i^-|$. Thus the best instance halves VS at every step resulting in an experiment sequence of length $O(log|VS|)$. Not every VS has a

halving instance; if none exists, we choose one that has the smallest value for E(i,VS).

Our strategy for DEG can now be stated simply as *Select the instance that minimizes* E(i,VS).

The time complexity of a generate-and-test implementation of DEG is $|VS||I|t$ where t is the time to compute the $\models$ relation. This computation is infeasible because the version space is very large even for simple concepts in C_1.

This naive method can be improved by using the fact that VS is partially ordered. The middle node w of the version space (the concept that is the root of half the nodes) is found, and then an instance which matches w — but matches no other concept descriptions more specific than w — is selected. However not all version spaces have middle nodes – very branchy partial orders still need $\Omega(|VS||I|t)$ amount of processing for the selection of the best instance.

Given the partial order on VS, we can try to generate the next best instance by using the boundary sets (S and G in [Mit78]) and the nature of the generalization mechanism (deduction). Estimation of E(i,VS) in the general case is very difficult without constructing the entire version space. Hence we have looked for sources of knowledge that reduce the size of the version space, allowing the optimal instance sequence to be constructed by the naive generation method in a smaller space.

V Exploiting the structure of the version space

We will now show how the generation of the best instance can be speeded up if we know some properties of the concept to be learned. An example of such a property is that the blocks world structure that is being learned, is a *stable* one. This is a domain-specific property by which the size of the VS can be reduced directly, without the use of instances in I. The learner simply prunes those descriptions that do not conform to its stability theory T.[1] DEG can then be used to learn the concept in the smaller VS.

A domain-independent property of the concept that can be used to make DEG more efficient is its factorability. The concept `red wedge` is factorable into `red` and `wedge`. All the concepts in our example VS are factorable into the colour component and the shape component. This is because colour and shape are independent relations in C_1. This suggests dividing the original learning problem into two independent learning problems: learning the shape and learning the colour. Two separate version spaces can be maintained, one for each component, and experiment design can be done by obtaining the best instance in each factor (by the method indicated in section 2.2). If credit or blame is assigned independently to each component of the instance, we say that *independent credit assignment (ICA)* is available. If the VS is factorable into k almost equal factors of size n, and if the induced factors in I are each of size m, then we have reduced a problem of size $n^k m^k t$ to k problems of size nmt, under ICA. This is clearly a significant computational gain.

The factors of the VS can be collapsed into singletons in either of two ways:

- In Parallel
  This is the optimal strategy to use if ICA is available. Each factor will be guaranteed to reduce in size by the maximal amount by DEG.
- In Series
  This is the basis for hierarchical learning as illustrated in Figure 4. Because the factors are logically independent, the order in which they are learned does not matter. If ICA is not available, this is the preferred strategy. The learner collapses one VS factor at a time. The negative instances generated will be near-misses[Win70].

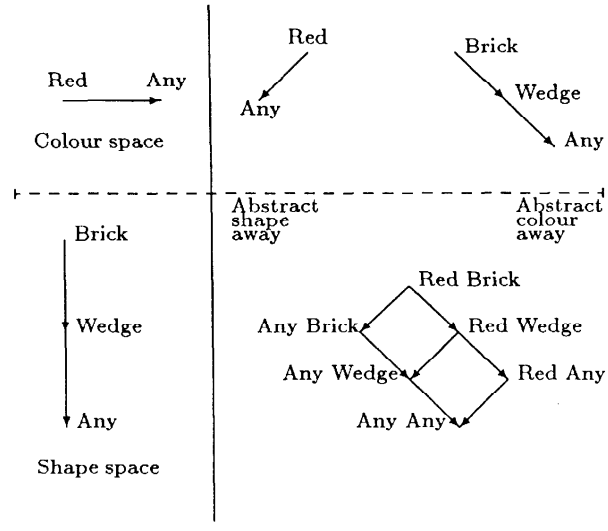

Figure 4

VI Formal analysis of factoring under ICA

We will relate the structural property of cartesian factorability of VS to the logical independence of parts of the concept being learned. We use graph theory to formalize the notion of cartesian factorability. We introduce the necessary definitions first:

Definition 1: A theory T is factorable iff $\exists\, T_1, T_2, \ldots, T_n$ such that

- $\cup_{i=1}^{n} T_i = \mathrm{T}$
- $\mathrm{T} - T_i \not\models T_i$

If each of the T_i is unfactorable, we call them the *irreducible factors* of T.

Definition 2: A partially ordered set VS of theories is factorable iff $\exists\, VS_1, VS_2, \ldots, VS_k$ such that

- VS $= \{\ vs_1 \wedge vs_2 \wedge \ldots \wedge vs_k \mid vs_1 \in VS_1, vs_2 \in VS_2, \ldots, vs_k \in VS_k\ \}$
  written as VS $= VS_1 \times VS_2 \times \ldots \times VS_k$
- $VS_1, VS_2, \ldots, VS_{i-1}, VS_{i+1}, \ldots, VS_k \not\models VS_i$, $1 \leq i \leq k$
- Each VS_i respects the partial order on VS, i.e.
  $(vs_1 \wedge vs_2 \wedge \ldots \wedge vs_i \wedge \ldots \wedge vs_n) \leq (vs_1 \wedge vs_2 \wedge \ldots \wedge vs'_i \wedge \ldots \wedge vs_n) \Rightarrow vs_i \leq_i vs'_i$

Again, if each of the VS_i's is unfactorable, they are called the irreducible factors of VS.

[1] However, after this pruning, the VS may no longer be representable in terms of its boundary sets.

Definition 3: If D_1 and D_2 are DAGs, then their *cartesian product* D (denoted as $D_1 \times D_2$) is defined as follows: The vertex set $V(D)$ is the cartesian product of the vertex sets of D_1 and D_2. The arc set of D, $A(D)$ = { (x,y) → (u,v) : (x = u and y → v ∈ $A(D_2)$) or (y = v and x → u ∈ $A(D_1)$) }. Clearly, this definition can be generalized to the product of k factors $D_1, D_2, \ldots, D_k$. A DAG D is called *cartesian-factorable* if there are two DAGs D_1 and D_2, both with $|V(D_i)| \geq 2$ such that $D = D_1 \times D_2$. A DAG with no nontrivial factorizations is called prime. Cartesian multiplication of DAGs is a commutative operation. Every finite DAG has a *unique* set of prime factors that can be found in polynomial time [Fei86].

The factors of a factorable concept can be generalized or specialized independently. This requires that C be factorable. C_1 in our example is presented in factored form. If C is finite, we can use the polynomial-time graph-factoring algorithm in [FHS85][Fei86] to make the factorization explicit. This gives us a way of discovering independence relations in the concept language.

Observation 1: If we construct all generalizations and specializations of a factorable theory using a factored C, the resulting (partially ordered) set of theories is factorable. The DAG that represents this set of theories is cartesian-factorable.
This means that the syntactic operation of graph factoring in Definition 3 corresponds to the semantic notion of logical factoring of the version space (Definition 2 above).

Observation 2: The initial version space of a factorable concept is a cartesian factorable DAG.
This is because the first positive instance is factorable and the version space is constructed as in Observation 1 above.

Observation 3: Under ICA, the update operations on the version space are guaranteed to preserve its factorability.
The update algorithm under ICA for the factored version space is:

for j = 1 to k do

if i_j is positive then replace VS_j by $VS_j{}^+$ else replace VS_j by $VS_j{}^-$

The updated unfactored version space is the product of the updated factors.

Observation 4: The best strategy for generating an instance when the version space is factorable, is to choose the best instance in each factor (i.e. the one that splits each factored space in half).

Because all hypotheses in VS are equally likely and all factors are logically independent, all hypotheses in VS_i are also equally likely. Thus the best strategy in each subspace is choosing an instance with the minimal E(i,VS).

Observation 5: We can compute E(i,VS) of every instance in the unfactored space from (I_1, $|VS_1{}^+|$, $|VS_1{}^-|$) and (I_2, $|VS_2{}^+|$, $|VS_2{}^-|$) tables.

Construction: Consider the instance i = $i_1 \wedge i_2$.
We have $|VS^+| = |VS_1{}^+| \cdot |VS_2{}^+|$ and $|VS^-| = |VS| - |VS^+|$.
We can then calculate E(i,VS) using $|VS^+|$ and $|VS^-|$. The table below shows this computation for our example. [2]

i	a	b	E	i_1	c	d	E_1	i_2	e	f	E_2
RB	6	0	6	R	2	0	2	B	3	0	3
RC	4	2	$\frac{10}{3}$	G	1	1	1	C	2	1	$\frac{5}{3}$
RP	2	4	$\frac{10}{3}$					P	1	2	$\frac{5}{3}$
GB	3	3	3								
GC	2	4	$\frac{10}{3}$								
GP	1	5	$\frac{13}{3}$								
Best i: GB				Best i_1: G				Best i_2: C or P			

This construction generalizes to the case where VS has more than 2 factors.

Observation 6: The best instance in the unfactored space is *not* the conjunction of the best instances in the factored spaces.

The nature of the feedback obtained from the teacher is different in the two cases. In the unfactored space, if **green brick** has been marked negative, then the version space is updated so that all three possibilities(**green** is OK but **brick** isn't, **green** isn't OK but **brick** is, **green** and **brick** are both not OK) are kept. In the factored space, because of ICA, we get more information per instance from the teacher: thus only one of the three possibilities above will be maintained.

VII Analysis for the non-ICA case

Usually, ICA is not available in the real world (e.g., digital circuit diagnosis), and the analysis is complicated by the fact that the learner has to do the credit assignment on negative examples by itself in order to update its factors. We now present a strategy for the learner under these conditions.

1. Generate the instance i that is the conjunction of all i_j's where each i_j is the best instance in each factor.
2. Ask teacher if i is positive or negative
3. If i is positive, replace every VS_j by VS_j+.
4. If i is negative,the learner needs to find which of the i_j's are negative: Let p = $p_1 \wedge p_2 \wedge \ldots p_j \wedge \ldots \wedge p_k$ be a *known* positive instance.
   - for j = 1 to k do
   - Ask teacher about $p_1 \wedge \ldots p_{j-1} \wedge i_j \wedge p_{j+1} \wedge \ldots \wedge p_k$. If it is positive, replace VS_j by VS_j+ else replace VS_j by VS_j-

Because each i_j is potentially faulty, the learner asks k questions to do the credit assignment. This credit assignment method corresponds closely to Winston's near-miss idea. A more sophisticated credit assignment strategy uses binary search. The learner replaces $\frac{k}{2}$ of the factors in a known positive instance. If that instance is labelled positive, it exonerates $\frac{k}{2}$ of the factors in one fell swoop. If it is labelled negative, further credit assignment instances need to be generated using the same strategy.

Observation 7: The version space update algorithm does not preserve factorability under non-ICA. The instances that the learner generates to do the credit assignment are guaranteed to restore factorability to the version space.

Without the credit assignment instances, under non-ICA, the updated version space will be a disjunction (disjoint union, in graph theoretic terms) of the $2^k - 1$ possible updates, where k is the number of factors. This graph is prime, even though each of its $2^k - 1$ components is factorable. The credit assignment instances seek to isolate that update and hence restore factorability to the version space.

[2] a is $|VS^+|$, b is $|VS^-|$, c is $|VS_1{}^+|$, d is $|VS_1{}^-|$, e is $|VS_2{}^+|$, f is $|VS_2{}^-|$

VIII Tradeoffs associated with factoring

The problem above points to a tradeoff associated with factoring: in general, the finer the factoring, the harder the experiment generator has to work on the credit assignment. This gives us a way of choosing how large k (the number of factors) should be, given that we don't have to factor down to irreducible factors. The formal cost comparison to be made is:

$$n^k m^k t + \mathrm{r}(\mathrm{a} + \mathrm{b}) > kmnt + \mathrm{r}(\mathrm{a} + \mathrm{kb}) + \mathrm{F}$$

where

1. a is the number of positive instances needed to learn the concept in the unfactored space,
2. b is the number of negative instances needed,
3. r is the cost of asking a question of the teacher.
4. F is the one-time cost of constructing the factored space.

This equation characterizes the conditions under which factoring is a good idea during experiment generation. Usually, the one-time cost of factoring is much smaller than the expected gain obtained by the use of factoring. In the ICA case, if the cost of asking questions is small, factoring up to irreducible factors is optimal. Factoring in the non-ICA case is a win exactly when the equation above holds: r will control how large k will be. Since r may vary from example to example, we use an average r in the above equation to compute k. The savings obtained by generating instances in the factored space are offset by the cost of asking the credit assignment questions (the cost of generating these is negligible). A judicious choice of k (which is constrained by the definition for factorability) will ensure that factoring leads to computational gains in the non-ICA case.

IX Implementation

The factoring method introduced in this paper has been used in the context of an implementation (called VS) of the generalized version space algorithm [Sub84]. VS is built on top of MRS, a logic based representation system. VS was tried out on several concept learning problems in the blocks world. The time for learning a concept increased exponentially with the number of conjuncts in it. Factoring instances and working with multiple version spaces proved to be a very powerful strategy. The size of the unfactored version space for Winston's arch problem[Win70] was approximately 3^9. VS used the cost formula in section 8 to determine the optimal number of factors(3 of size 27 each), and the arch was learned using 9 instances which were generated by the experiment generator in VS.

The definition of factoring presented here hinges on the conjunctive factorability of the concepts in the version space. We have a similar notion of factoring that applies to disjunctions, except that we use disjoint union (instead of cartesian product) as the composition operator. We have used this in the design of experiments in digital circuit diagnosis. Only constant factor speedups have been obtained in this case. These two definitions can be combined to factor more complex version spaces.

X Conclusions

In this paper we proposed *factoring* as a technique for generating discrimination experiments efficiently. We analyzed the applicability conditions for this technique and presented optimal strategies for its use under a varying set of assumptions. The effectiveness of this method has been experimentally verified on several examples.

This work gives computational justification for some well-known maxims in the design of concept languages. The independence between relations should be stated explicitly so that they can be directly used to factor concepts. Also, the choice of relations should be such that they reflect independences in the world.

The results obtained here can be extended to handle partial independence relations, in which some communication between the factors is needed. The implementation in [Sub84] deals with the sharing of variable bindings between factors. Further work includes relaxing the uniform probability assumptions made in the analysis of factoring and building a logical framework in which intelligent experiment generation strategies can be derived.

Acknowledgements

We thank Professor Genesereth and Stuart Russell for their valuable criticisms on an early draft of this paper. David Wilkins, Haym Hirsh, Chris Fraley and members of GRAIL provided useful feedback. Thanks also to the anonymous reviewers for their comments. The experiment generator was implemented on the SUMEX computer facilities at the Knowledge Systems Laboratory, Stanford University.

References

[DB83] T.G. Dietterich, and B.G. Buchanan. "The Role of Experimentation in Theory Formation", in *Proceedings of the International Workshop on Machine Learning*, Univ. of Illinois at Urbana-Champaign, pp. 147-155, June 1983.

[FHS85] J. Feigenbaum, J. Hershberger, and A.A Schäffer. "A Polynomial-time Algorithm for Finding the Prime Factors of Cartesian-Product Graphs", *Discrete Applied Mathematics*, 12,2(1985), 123-138.

[Fei86] J. Feigenbaum. "Directed Cartesian-Product Graphs have Unique Prime Factors that Can be Found in Polynomial Time", to appear in *Discrete Applied Mathematics*.

[Mit78] T. Mitchell. *Version Spaces: An Approach to Concept Learning*, Ph.D. dissertation, Stanford University, December 1978.

[Mit83] T. Mitchell, P. Utgoff, R. Banerji. "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics", in *Machine Learning I*, Mitchell, Michalski, Carbonell and Mitchell(eds.), Tioga Publishing Company, 163-189.

[Sub84] D. Subramanian. "Experiment Generation with Version Spaces", HPP-84-45, December 1984, revised March 1986.

[Win70] P.H. Winston, "Learning Structural Descriptions from Examples", *The Psychology of Computer Vision*, Winston, P.H. (ed.), McGraw Hill, NY, 1975.