Learning Concepts in Parallel Based upon the Strategy of Version Space

Tzung-Pei Hong and Shian-Shyong Tseng, *Member, IEEE*

***Abstract*— In this paper, we have attempted to apply the technique of parallel processing to concept learning. A parallel version-space learning algorithm based upon the principle of divide-and-conquer is proposed. Its time complexity is analyzed to be $O(k \log_2 n)$ with n processors, where n is the number of given training instances and k is a coefficient depending on application domains. For a bounded number of processors in the real situations, a modified parallel learning algorithm is then proposed. Experimental results are then performed on a real learning problem, showing our parallel learning algorithm works and being quite consistent with results of theoretic analysis. We have finally concluded that when the number of training instances is large, it is worth learning in parallel because of its faster execution.**

***Index Terms*—Divide-and-conquer, generalization process, hypothesis, parallel learning, specialization process, training instance, version space**

I. Introduction

LEARNING general concepts from a set of training instances has become increasingly important for artificial intelligence researchers in constructing knowledge-based systems [2], [3], [7], [8], [28]. This problem has been studied by many researchers over the last two decades; many approaches have been proposed to solve it [17], [18]. Learning strategy adopted can be divided into two classes: data-driven strategy and model-driven strategy [5], [21], [22]. Data-driven strategy processes input examples one at a time, gradually generalizing the current set of descriptions until a final conjunctive hypothesis is computed. Therefore, it processes in a bottom-up way. On the other hand, model-driven strategy searches a set of possible generalizations in an attempt to find a few best hypotheses satisfying certain requirements by considering an entire set of training instances as a whole. So, it processes in a top-down manner.

No matter which strategy is adopted, its efficiency is limited by its learning speed. Because of the dramatic increase in computing power and the concomitant decrease in computing cost over last decade, learning from examples by parallel processing has become a feasible way for conquering the low-speed problem in learning within a single processor [11], [13], [14].

In this paper, one famous data-driven learning strategy, called "version space" [19]–[22] is adopted as our strategy for parallel learning, because its characteristic of not checking past training instances makes independent processing among different processors possible. We then discuss the feasibility of parallel learning on the strategy of "version space" and propose a parallel learning algorithm that can be accomplished in $O(k \log_2 n)$ (where n is the number of given training instances and k is a coefficient depending on application domains). Proof of correctness in the proposed parallel learning algorithm is also given. This algorithm is further modified for practical restriction to a bounded number of processors. Experiments on the Iris Learning Problem [6], [10] finally ensure the validity of our parallel learning algorithm.

This paper is organized as follows. The learning problem considered in this paper is formally defined in Section II. The version space learning strategy is introduced in Section III. A parallel learning model and a parallel learning algorithm are proposed in Section IV. This is followed by analysis of time complexity in Section V. A modified parallel learning algorithm for a bounded number of processors is then proposed in Section VI. Some experiments to verify effectiveness of our parallel learning algorithm are made in Section VII. Conclusion and future work are finally summarized in Section VIII.

II. Learning Problem

Before describing the learning problem, some terminology should first be defined. An *instance space* is a set of instances that can be legally described by a given instance language. Instance spaces can be divided into two classes: *attribute-based* instance spaces and *structured* instance spaces. In an attribute-based instance space, each instance can be represented by one or several attributes. Each attribute may be nominal, linear, or tree-structured [30] (Fig. 1). For example, instance "color = red, weight = 1, shape = oval" is attribute-based. Attribute-based instance spaces are of the main concern here.

A *hypothesis space* is a set of hypotheses that can be legally described by a concept description language (generalization language). The most prevalent form of a hypothesis space is restriction to concepts that can be expressed only in conjunc-

Manuscript received May 20, 1991; revised June 18, 1993. This work was supported by the National Science Council of the Republic of China under Contract NSC81-0408-E009-16.

T.-P. Hong is with the Department of Computer Science, Chung-Hua Polytechnic Institute, Hsinchu, Taiwan 30067 Republic of China; e-mail: tphong@chpi.edu.tw.

S.-S. Tseng is with the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu, Taiwan 30050 Republic of China; e-mail: sstseng@cis.nctu.edu.tw.

IEEE Log Number 9213314.

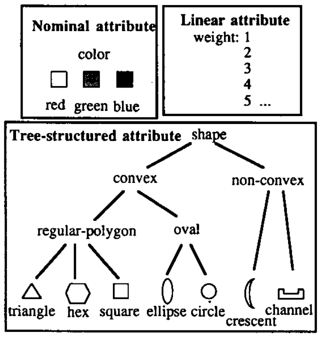

Fig. 1. Attribute domain.

tive forms. For example, hypothesis "color = red and shape = convex" are in conjunctive forms. Because some unsolved problems in learning disjunctive concepts based on version space strategy [4], [20] still exist, the hypothesis spaces here are restricted to conjunctive forms.

Given an instance space and a hypothesis space, a set of predicates is still required to test whether a given hypothesis matches a given instance (i.e., whether the given instance is contained in the instance set corresponding to the given hypothesis). For example, in a tree-structured attribute domain, one matching predicate is a predecessor-successor relation in the hierarchy tree. In other words, a hypothesis matches an instance if the instance is a successor of the hypothesis in the hierarchy tree.

Two partial ordering relations, called "*more-specific-than*" ($\leq$) and "*more-general-than*" ($\geq$) exist in the hypothesis space. Hypothesis $A \leq$ hypothesis B (or $B \geq A$) iff each instance contained in A is also contained in B. Note that these two relations are *reflexive*. That is, $A \leq A$ and $A \geq A$. These partial ordering relations are important because they provide a powerful basis for organizing the search through the hypothesis space [20].

A hypothesis A is a least general generalization (lgg) [24], [25] of two hypotheses B and C (1) if $A \geq B$ and $A \geq C$; and (2) if another hypothesis $A' \geq B$ and $A' \geq C$, then $\neg(A \geq A')$. Similarly, a hypothesis A is a least specific specification (lss) of two hypotheses B and C iff (1) $A \leq B$ and $A \leq C$; and (2) if another hypothesis $A' \leq B$ and $A' \leq C$, then $\neg(A \leq A')$.

The learning problem to be solved in this paper can now be defined as in [20].

Given the following information:

1) *instance space*,
2) *hypothesis space*,
3) *a set of predicates to test whether a given hypothesis matches a given instance*, and
4) *a set of positive and negative training instances of a target concept to be learned*,

determine *one or several hypotheses in conjunctive forms, each of which is consistent with the presented training instances.*

The term "*consistent*" means that this hypothesis matches (includes) all given positive training instances and matches no (excludes) given negative ones.

Example 1: Consider the learning problem of classifying examples belonging to different kinds of iris flowers [6], [10]. Assume that the following information is given.

1) *Instance Space:* Each training instance is described by four attributes—Sepal Width (S.W), Sepal Length (S.L), Petal Width (P.W), and Petal Length (P.L). Units for all the four attributes are centermeter, measured to the nearest millimeter.
2) *Hypothesis Space:* Each legal hypothesis is restricted to be conjunctions of form $a \leq X < b$ (for each attribute X), where a and b are limited to multiples of 8 mm.
3) *Matching Predicates:* A hypothesis H matches a training instance I if and only if the value of each attribute in I is within the range of the corresponding attribute in H.
4) *Training Set:* Positive instances—(S.L = 5.1, S.W = 3.5, P.L = 1.4, P.W = 0.2), and (S.L = 4.3, S.W = 3.0, P.L = 1.1, P.W = 0.1), and negative instances—(S.L = 7.0, S.W = 3.3, P.L = 4.7, P.W = 1.4).

According to above information, any one hypothesis more general than ($4.0 \leq$ S.L < 5.6, $2.4 \leq$ S.W < 4.0, $0.8 \leq$ P.L < 1.6, $0.0 \leq$ P.W < 0.8), and more specific than ($0.0 \leq$ S.L < 6.4, $0.0 \leq$ S.W $< \infty$, $0.0 \leq$ P.L $< \infty$, $0.0 \leq$ P.W $< \infty$), ($0.0 \leq$ S.L $< \infty$, $0.0 \leq$ S.W $< \infty$, $0.0 \leq$ P.L < 4.0, $0.0 \leq$ P.W $< \infty$), or ($0.0 \leq$ S.L $< \infty$, $0.0 \leq$ S.W $< \infty$, $0.0 \leq$ P.L $< \infty$, $0.0 \leq$ P.W < 0.8) is desired. Methods in achieving the boundaries are discussed in the next section.

III. Overview of Version Space Learning Strategy

Version space learning strategy [19], [20] was proposed by Mitchell in 1978, having been applied successfully in some systems such as Meta-DENDRAL [3] and LEX [23]. The term "*version space*" is used to represent all legal hypotheses describable within a given concept description language and consistent with all observed training instances. A version space can be represented by two sets of hypotheses: set S and dual set G, defined as follows.

$S = \{s \mid s$ is a hypothesis consistent with observed instances. No other hypothesis exists that is both more specific than s and consistent with observed instances$\}$;

$G = \{g \mid g$ is a hypothesis consistent with observed instances. No other hypothesis exists that is both more general than g and consistent with observed instances$\}$.

Sets S and G together precisely delimit the version space, and each hypothesis in the version space is both more general than some hypothesis in S and more specific than some hypothesis in G. When a new positive training instance appears, set S is generalized to include this training instance; when a new negative training instance appears, set G is specialized to exclude this training instance. An example is given below to clearly explain version space strategy.

Example 2: For the learning problem given in Example 1, learning process by *version space* strategy is shown in Fig. 2. In the newly formed version space by Step 3 (Fig. 2), ($7.2 \leq$ S.L $< \infty$, $0.0 \leq$ S.W $< \infty$, $0.0 \leq$ P.L $< \infty$, $0.0 \leq$ P.W $< \infty$) in G is discarded and then is not included in the newly formed version space, because there is no hypothesis more specific than it and more general than hypothesis ($4.0 \leq$ S.L

1. **(S.L=5.1, S.W=3.5, P.L=1.4, P.W=0.2)** *Positive training instance*

   S: (4.8≤S.L<5.6, 3.2≤S.W<4.0, 0.8≤P.L<1.6, 0.0≤P.W<0.8)

   G: (0.0≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<∞)

2. **(S.L=4.3, S.W=3.0, P.L=1.1, P.W=0.1)** *Positive training instance*

   S: (4.0≤S.L<5.6, 2.4≤S.W<4.0, 0.8≤P.L<1.6, 0.0≤P.W<0.8)

   G: (0.0≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<∞)

3. **(S.L=7.0, S.W=3.3, P.L=4.7, P.W=1.4)** *Negative training instance*

   S: (4.0≤S.L<5.6, 3.2≤S.W<4.0, 0.8≤P.L<1.6, 0.0≤P.W<0.8)

   G: (0.0≤S.L<6.4, 0.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<∞)

   (7.2≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<∞) Discarded

   (0.0≤S.L<∞, 0.0≤S.W<3.2, 0.0≤P.L<∞, 0.0≤P.W<∞) Discarded

   (0.0≤S.L<∞, 4.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<∞) Discarded

   (0.0≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<4.0, 0.0≤P.W<∞)

   (0.0≤S.L<∞, 0.0≤S.W<∞, 4.8≤P.L<∞, 0.0≤P.W<∞) Discarded

   (0.0≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<∞, 0.0≤P.W<0.8)

   (0.0≤S.L<∞, 0.0≤S.W<∞, 0.0≤P.L<∞, 1.6≤P.W<∞) Discarded

Fig. 2. Illustration for the version space strategy.

$< 5.6, 3.2 \leq$ S.W $< 4.0, 0.8 \leq$ P.L $< 1.6, 0.0 \leq$ P.W $< 0.8)$ in S.

Lemma 1: In a nonempty version space, each hypothesis in S/G must be more specific/general than at least one hypothesis in G/S.

Proof: Assume some hypothesis s in S is not more specific than any hypothesis in G. According to the definition of version space, each hypothesis in the version space V must be more specific than some hypothesis in G and more general than some hypothesis in S. s is then not in V, implying that s is not in S. A contradiction arises. Each hypothesis in S must then be more specific than at least one hypothesis in G. Similar arguments can be given for G. □

In parallel processing, coping with the synchronization problem is difficult if dependency among different processors is heavy [1], [26]. Because of the characteristic of not checking past training instances, the strategy of version space is suitable for rocessing in parallel.

IV. Learning Concepts in Parallel

By applying similar idea used in DADO [9], [29], a parallel learning model adopted is shown in Fig. 3. Each circle represents a training instance, and each rectangle represents a processing element.

All processing elements on the same level of the tree can work concurrently and in parallel. Learning process starts from the bottom rectangular level of the tree. At the beginning, each processing element inputs two separate training instances, finds sets S and G, which are defined in version space strategy, and forms a version space as output. Each processing element lying on one level higher than the previous one then inputs two version spaces, processing them in order to form an equivalent version space as output. This process is repeated along with the tree bottom-up until a final version space is obtained.

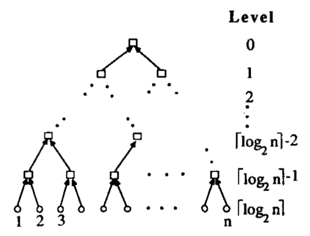

Fig. 3. Bottom-up processing in a binary tree.

A. Merging Two Version Spaces

The following lemma shows version space strategy allowing a convenient, consistent method for merging several sets of hypotheses generated from distinct training data sets [20].

Lemma 2: Intersection of two version spaces formed from two sets of training instances yields the version space consistent with the union of these two training sets.

Proof: Let V_1 and V_2 denote two original version spaces; I_1 and I_2 denote corresponding training sets. Since each hypothesis in V_1 matches all positive training instances and no negative training instances in I_1, and because each one in V_2 matches all positive training instances and no negative training instances in I_2, a hypothesis lying in both V_1 and V_2 must match all positive training instances and no negative training instances in I_1 and I_2. Intersection of V_1 and V_2 must then be consistent with the union of I_1 and I_2. □

Corollary 1: Let intersection of two version spaces, formed from two given training sets I_1 and I_2 with $I = I_1 \cup I_2$, be V. Intersection of two other version spaces, formed from two training sets I_3 and I_4 with $I = I_3 \cup I_4$, is then still V.

The following two lemmas can be used to find intersection of two version spaces.

Lemma 3: For each hypothesis s/g in S/G of the newly formed version space V from V_1 and V_2, there must exist some hypothesis s_1/g_1 in S_1/G_2 and s_2/g_2 in S_2/G_2 such that s/g is a lgg/lss of s_1/g_1 and s_2/g_2.

Proof: Without loss of generality, we need to prove only the case for a hypothesis s in S. According to definition of S, no other hypothesis exists more specific than s in V. Since V is the intersection of V_1 and V_2, s must be more general than some hypothesis s_1 in S_1 and some s_2 in S_2. Furthermore, s must be a lgg of both s_1 and s_2; otherwise, there exists a hypothesis in V that is more specific than s, contradicting the fact s is in S. □

Inverse statement of Lemma 3 is not always true. A lgg/lss of some s_1/g_1 in S_1/G_1 and some s_2/g_2 in S_2/G_2 is not necessarily in S/G. It may be subsumed by some other hypotheses in the same set or may contradict hypotheses in the dual set.

Lemma 4: For hypothesis s/g that is a lgg/lss of some hypothesis s_1/g_1 in S_1/G_1 and s_2/g_2 in S_2/G_2, and another hypothesis s'/g' that is a lgg/lss of another hypothesis s_1'/g_1' in S_1/G_1 and s_2'/g_2' in S_2/G_2, if s/g is more specific/general than s'/g', then s'/g' will not be in S/G.

Proof: Without loss of generality, only the case for S needs to be proven. If s is in S, s' will not then be in S, according to the definition of S. If s is not in S, it cannot then find any hypothesis in G more general than it, according to Lemma 1. Since s is more specific than s', s' cannot find any hypothesis in G more general than it neither. s' cannot be in S is implied. □

According to Lemma 3, the set (named S') that contains desired set S can be found by taking each hypothesis in S_1 and each in S_2 to generate the lgg's of the pair of chosen hypotheses. A *Generalization process* is defined here as the process of finding the lgg's of two chosen hypotheses. A *Specialization process* is similarly defined as the process of finding the lss's of two chosen hypotheses. Besides, Lemma 4 introduces an additional processing, *redundancy and subsumption checking*, in excluding the redundant and not least general hypotheses in S'. The process is similar for set G. After redundancy and subsumption checking, another type of checking called *contradiction checking* needs to be performed in discarding the hypotheses in S/G that is not more specific/general than any hypothesis in G/S due to Lemma 1.

Generally assume that two version spaces V_1 and V_2 exist, where corresponding S_1 contains a_1 hypotheses and corresponding S_2 contains a_2 hypotheses. That is, we define the following.

S_1: [$hypothesis_{11}$, $hypothesis_{12}, \cdots$, $hypothesis_{1a_1}$].
S_2: [$hypothesis_{21}$, $hypothesis_{22}$, $\cdots$, $hypothesis_{2a_2}$].

The process of merging S_1 and S_2 into an equivalent S is the Cartesian product of S_1 and S_2, denoted as $S_1 \times S_2$; restated, the lgg's of each hypothesis in S_1 and each hypothesis in S_2 are obtained. Redundancy, subsumption, and contradiction checking, meanwhile, should be done among newly formed hypotheses in S and G. Merging of G sets can be processed in a similar way.

B. Parallel Learning Algorithm

Parallel algorithm for concept learning can be outlined below from the above discussion. This is basically based upon the principle of divide-and-conquer [16], [27].

Parallel Learning Algorithm:

INPUT: A set I of n training instances.

OUTPUT: A version space V with sets S and G consistent with the set I.

STEP 1: Divide I into I_1 and I_2. The sizes of I_1 and I_2 are equal. (If the size of I is not even, add a virtual training instance to I to make the size even.) Besides, the distributions of positive and negative training instances in I_1 and I_2 are as equal as possible.

STEP 2: Recursively and in parallel apply the algorithm to find the version space V_1 for I_1 and V_2 for I_2, respectively.

STEP 3: Merge the two version spaces V_1 and V_2 into an equivalent version space V.

The version space for a virtual training instance here is the universal version space with S equal to the most specific hypothesis and G equal to the most general hypothesis in the hypothesis space. From Corollary 1, any partition of I will not affect correctness of the final version space. It indeed affects intermediate results, however, and then influences learning speed. By interleaving positive and negative training instances as much as possible, illegal hypotheses in S and G can be removed as early as possible. The needed learning time is then shorter. This is illustrated by the experiments in Section VII.

Each positive training instance can initially be viewed as a version space with set S containing only the instance itself; each negative training instance can be viewed as a version space with set G excluding only the instance itself. The merging algorithm is then described as follows:

Version Space Merging Algorithm:

INPUT: Two version spaces V_1 with S_1, G_1, and V_2 with S_2, G_2.

OUTPUT: An equivalent version space V with S and G.

STEP 1: Initialize both sets S and G to be ϕ.

STEP 2: Take a hypothesis in S_1 and a hypothesis in S_2 (in an order of (1, 1), (1, 2), $\cdots$, (1, $|S_2|$), (2, 1), (2, 2), $\cdots$, (2, $|S_2|$), $\cdots$, ($|S_1|$, 1), ($|S_1|$, 2), $\cdots$($|S_1|$, $|S_2|$), where $|X|$ denotes cardinality of set X) to perform a generalization process. Set newly formed hypotheses to be S'. Check S' with S against redundancy and subsumption with three cases possibly existing.

Case 1. If a hypothesis s' in S' is more general than some hypothesis s in S, discard s' in S'.

Case 2. If a hypothesis s' in S' is more specific than some hypothesis s in S, discard s and add s' to set S.

Case 3. Otherwise, add s' to set S.

STEP 3: Repeat STEP 2 until each hypothesis in S_1 is processed with each hypothesis in S_2.

STEP 4: Take a hypothesis in G_1 and a hypothesis in G_2 (in an order of (1, 1), (1, 2), $\cdots$, (1, $|G_2|$), (2, 1), (2, 2), $\cdots$, (2, $|G_2|$), $\cdots$, ($|G_1|$, 1), ($|G_1|$, 2), $\cdots$($|G_1|$, $|G_2|$)) to perform a specialization process. Set newly formed hypotheses to be G'. Check G' with G against redundancy and subsumption with three cases possibly existing.

Case 1. If a hypothesis g' in G' is more specific than some hypothesis g in G, discard g' in G'.

Case 2. If a hypothesis g' in G' is more general than some hypothesis g in G, discard g and add g' to set G.

Case 3. Otherwise, add g' to set G.

STEP 5: Repeat STEP 4 until each hypothesis in G_1 is processed with each hypothesis in G_2.

STEP 6: Take a hypothesis s in S and a hypothesis g in G (in an order of (1, 1), (1, 2), $\cdots$, (1, $|G|$), (2, 1), (2, 2), $\cdots$, (2, $|G|$), $\cdots$, ($|S|$, 1), ($|S|$, 2), $\cdots$($|S|$, $|G|$)). Check s with g against contradiction with two cases possibly existing.

Case 1. If g is not more general than s, mark s and g.

Case 2. Otherwise, do nothing.

STEP 7: Repeat STEP 6 until each hypothesis in S is processed with each hypothesis in G.

STEP 8: Discard those hypotheses in S with $|G|$ marks and those in G with $|S|$ marks.

After execution of Step 8, the desired version space is obtained. Note in Case 1 of Step 2/Step 4, the process includes checking of redundancy. This is because more-specific-than and more-general-than relations have the reflexive property. Correctness of our algorithm is shown below.

Theorem 1: The version space obtained by the version space merging algorithm here is the intersection of two given original version spaces.

Proof: Let S'' denote set S obtained after Step 3, and let G" denote set G obtained after Step 5 for the sake of clarity. Because of Step 2, in generated hypotheses, only the least general ones are kept, such that each hypothesis in S'' is not more general than the others. According to Lemmas 3 and 4, S'' is a superset of the final S. S'' can then be divided into two disjoint sets: the final S and S''-S. It implies that any hypothesis in S''-S is not in the final version space and is not more specific than any hypothesis in G by Lemma 1. Otherwise, some hypothesis in S''-S must be in S. Set G'' after STEP 5 can similarly be divided into the final set G and G''-G. In Step 6 to Step 8, S'' and G'' are checked for contradiction and G''-G and S''-S will be discarded after Step 8. After Step 8, the final sets G and S are then desirable. □

It can easily be seen that Steps 2 and 3 can be performed in parallel; Steps 4 and 5 and Steps 6 to 8 are also the same cases. The merging algorithm described above can then be further parallelized if q free processors are available.

Parallel Version Space Merging Algorithm:

INPUT: Two version spaces V_1 with S_1, G_1, and V_2 with S_2, G_2.

OUTPUT: An equivalent version space V with S and G.

PSTEP 1: In each processor P_i, initialize both sets SP_i and GP_i to be ϕ.

PSTEP 2: Divide and assign $|S_1|$ hypotheses of S_1 (without loss of generality, assume $|S_1| \geq |S_2|$) as equally as possible onto the available q processors. Refer to the set in P_i as SP_{1i}. In each processor P_i, take a hypothesis in SP_{1i} and a hypothesis in S_2 to perform a generalization process. Set newly formed hypotheses to be SP_i'. Check SP_i' with SP_i against redundancy and subsumption with three cases possibly existing.

Case 1. If a hypothesis s' in SP_i' is more general than some hypothesis s in SP_i, discard s' in SP_i'.

Case 2. If a hypothesis s' in SP_i' is more specific than some hypothesis s in SP_i, discard s and add s' to set SP_i.

Case 3. Otherwise, add s' to set SP_i.

Repeat this step until each hypothesis in SP_{1i} is processed with each hypothesis in S_2.

PSTEP 3: Pairwise and in parallel, merge-check SP_1, SP_2, $\cdots$, SP_q against interprocessor redundancy and subsumption by the bottom-up way (Fig. 4). In each intermediate processor for managing sets SP_i and SP_j, check each hypothesis s' in SP_i with SP_j, with three cases possibly existing.

Case 1. If s' is more general than some hypothesis s in SP_j, discard s' in SP_i.

Case 2. If s' is more specific than some hypothesis s in SP_j, discard s and add s' to set SP_j.

Case 3. Otherwise, add s' to set SP_j.

Output SP_j upward for further merge-check. Refer to the set output at Level 0 as S'.

PSTEP 4: Divide and assign $|G_1|$ hypotheses of G_1 (without loss of generality, assume $|G_1| \geq |G_2|$) as equally as possible onto the q available processors. Refer to the set in P_i as GP_{1i}. In each processor P_i, take a hypothesis in GP_{1i} and a hypothesis in G_2 to perform a specialization process. Set newly formed hypotheses to be GP_i'. Check GP_i' with GP_i against redundancy and subsumption, with three cases possibly existing.

Case 1. If a hypothesis g' in GP_i' is more specific than some hypothesis g in GP_i, discard g' in GP_i'.

Case 2. If a hypothesis g' in GP_i' is more general than some hypothesis g in GP_i, discard g and add g' to set GP_i.

Case 3. Otherwise, add g' to set GP_i;

Repeat this step until each hypothesis in GP_{1i} is processed with each hypothesis in G_2.

PSTEP 5: Pairwise and in parallel, merge-check GP_1, GP_2, $\cdots$, GP_q against interprocessor redundancy and subsumption by the bottom-up way (Fig. 4). In each intermediate processor for managing sets GP_i and GP_j, check each hypothesis g' in GP_i with GP_j with three cases possibly existing.

Case 1. If g' is more specific than some hypothesis g in GP_j, discard g' in GP_i;

Case 2. If g' is more general than some hypothesis g in GP_j, discard g and add g' to set GP_j.

Case 3. Otherwise, add g' to set GP_j.

Output GP_j upward for further merge-check. Refer to the set output at Level 0 as G'.

PSTEP 6: Divide and assign $|G'|$ hypotheses of G' (without loss of generality, assume $|G'| \geq |S'|$) as equally as possible onto the available q processors. Refer to the set in P_i as G_i'. In each processor P_i, check each hypothesis s in S' with each hypothesis g in G_i' against contradiction with two cases possibly existing.

Case 1. If g is not more general than s, mark s and g.

Case 2. Otherwise, do nothing.

Repeat this step until each hypothesis in S' is processed with each hypothesis in G_i'. Discard those hypotheses in G_i' with $|S'|$ marks. Collect all hypotheses in G_i''s as G.

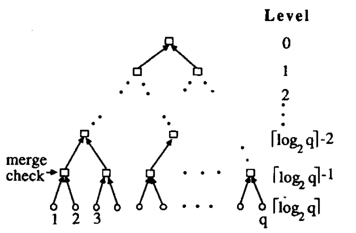

Fig. 4. Illustration for parallel merging.

PSTEP 7: Pairwise and in parallel, count the total number of marks of each hypothesis s in S' by the bottom-up way (Fig. 4). Discard the hypotheses of S' with $|G'|$ marks, referring to the altered set as S.

Theorem 2: The version space obtained by the parallel version space merging algorithm here is the intersection of two given original version spaces.

Proof: It can easily be proven that each intermediate result at end of PSTEP 3, PSTEP 5, and PSTEP 7 of the parallel merging algorithm is, respectively, the same as that at end of STEP 3, STEP 5, and STEP 8 of the sequential merging algorithm. After PSTEP 7, sets S and G then constitute the desired version space. □

Theorem 3: The parallel version space learning algorithm is correct.

Proof: According to Theorems 1 and 2, correctness of our parallel learning algorithm can easily be proven by induction. □

Example 3: For the learning problem described in Example 1, assume the training set is given as follows:

positive instances: (5.1, 3.5, 1.4, 0.2), (4.3, 3.0, 1.1, 0.1), (5.7, 4.4, 1.5, 0.4), and (4.8, 3.4, 1.9, 0.2),
negative instances: (7.0, 3.3, 4.7, 1.4), (6.4, 2.7, 5.3, 1.9), (5.0, 2.7, 3.9, 1.4), and (5.5, 2.8, 4.9, 2.0).

Parallel learning process by version space strategy is shown in Fig. 5; each notation $a \sim b$ represents $a \leq X < b$ for value X of some attribute.

V. Time Complexity Analysis

In this section, time complexities of our parallel learning algorithm and the sequential learning algorithm proposed by Mitchell [20], [22] are analyzed and compared. For the purpose of having a criterion about processing time, a unit operation is defined below.

Definition:

Unit operation: A specialization or generalization process from any two chosen hypotheses is defined as a *unit operation.*

The unit operation defined above, most likely taken as the time unit in Mitchell's sequential learning algorithm analysis [20], is from here on considered as a basic time unit of processing.

For the merging algorithm proposed in Section IV-B, processing time includes the required number of specialization/generalization processes and three types of checking.

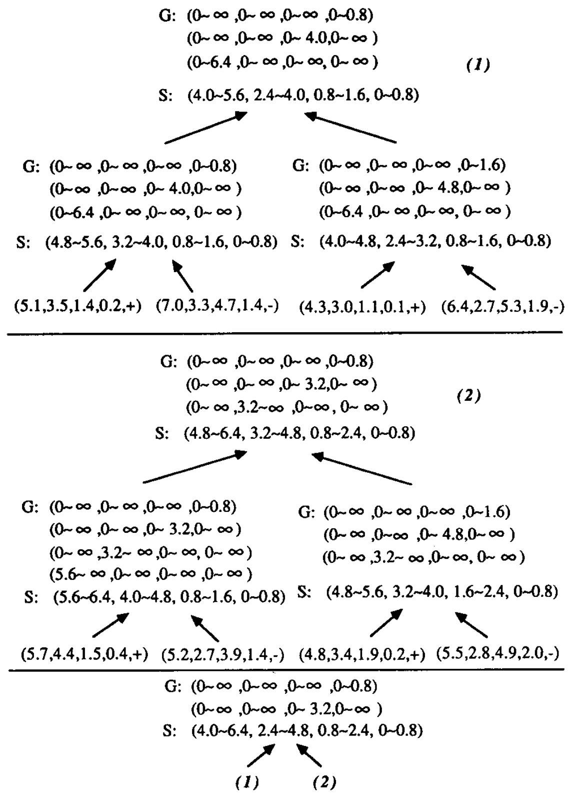

Fig. 5. Illustration of parallel version space strategy.

Checking is used for examination of redundancy, subsumption, and contradiction among hypotheses in sets S and G. Checking between any two hypotheses can be finished within a unit operation, because it is simpler than a specialization process or a generalization process.

We now focus on time complexity analysis of our parallel learning algorithm. Let $T_q(n)$ denote time complexity of our parallel learning algorithm in dealing with n training instances with q processors available (assume $n \leq q$; the case for $n > q$ is discussed in Section VI). Let $M_q(n)$ denote time complexity of merging two version spaces V_1 and V_2, each of which is formed from $n/2$ training instances, into an equivalent version space with q processors available. In this parallel learning environment, the following formula holds:

$$T_q(n) = T_{q/2}(n/2) + M_q(n). \tag{1}$$

In analyzing time complexity of $M_q(n)$, let $s_{\max}$ and $g_{\max}$, respectively, denote the maximum numbers of hypotheses in sets S and G appearing in the whole learning process. For the parallel merging algorithm outlined in Section IV, the time complexity of each step is listed in Table I. Therefore, we have the following:

$$\begin{aligned} M_q(n) = O(&s_{\max}^3/q + s_{\max}^2\lceil\log_2 q\rceil + g_{\max}^3/q \\ &+ g_{\max}^2\lceil\log_2 q\rceil) + s_{\max} \times g_{\max}/q \\ &+ s_{\max}\lceil\log_2 q\rceil) \text{ for } q < s_{\max} \text{ and } g_{\max}, \end{aligned} \tag{2}$$

TABLE I
TIME COMPLEXITY IN THE PARALLEL MERGING ALGORITHM

	Time Complexity	
PSTEP 1	O(1)	
PSTEP 2	$O(s_{max}^3/q)$ if $q < s_{max}$	$O(s_{max}^2)$ if $q \geq s_{max}$
PSTEP 3	$O(s_{max}^2 \lceil \log_2 q \rceil)$ if $q < s_{max}$	$O(s_{max}^2 \lceil \log_2 s_{max} \rceil)$ if $q \geq s_{max}$
PSTEP 4	$O(g_{max}^3/q)$ if $q < g_{max}$	$O(g_{max}^2)$ if $q \geq g_{max}$
PSTEP 5	$O(g_{max}^2 \lceil \log_2 q \rceil)$ if $q < g_{max}$	$O(g_{max}^2 \lceil \log_2 g_{max} \rceil)$ if $q \geq g_{max}$
PSTEP 6	$O(s_{max} g_{max}/q)$ if $q < g_{max}$	$O(s_{max})$ if $q \geq g_{max}$
PSTEP 7	$O(s_{max} \lceil \log_2 q \rceil)$ if $q < g_{max}$	$O(s_{max} \lceil \log_2 g_{max} \rceil)$ if $q \geq g_{max}$

and

$$\begin{aligned} M_q(n) = O(&s_{max}^2 + s_{max}^2 \lceil \log_2 s_{max} \rceil + g_{max}^2 \\ &+ g_{max}^2 \lceil \log_2 g_{max} \rceil + s_{max} + s_{max} \lceil \log_2 g_{max} \rceil) \\ &\text{for } q \geq s_{max} \text{ and } g_{max}. \end{aligned} \tag{3}$$

It can be seen from (2) and (3) that $M_q(i) = M_q(j)$ for any i, j. This is because they are analyzed by s_{max} and g_{max}, which are the maximum numbers of hypotheses in sets S and G appearing in the whole hypothesis space.

Assume that n processors are available with each training instance initially being located in an individual processor. Only two processors are then available for each merging process in Level $\lceil \log_2 n \rceil - 1$ and four processors are available in Level $\lceil \log_2 n \rceil - 2$. In general, 2^k processors are available in Level $\lceil \log_2 n \rceil - k$. In this case, time complexity is arrived at by calculating as follows:

$$\begin{aligned} T_n(n) &= T_{n/2}(n/2) + M_n(n) \\ &= \sum_{k=1}^{\lceil \log_2 n \rceil} M_{2^k}(2^k) \\ &= O((s_{max}^2 + s_{max}^2 \lceil \log_2 s_{max} \rceil + g_{max}^2 \\ &\quad + g_{max}^2 \lceil \log_2 g_{max} \rceil + s_{max} + s_{max} \lceil \log_2 g_{max} \rceil) \\ &\quad \times \lceil \log_2 n \rceil + s_{max}^3 + g_{max}^3 + s_{max} \times g_{max} - s_{max}^2 \\ &\quad - s_{max}^2 \lceil \log_2 s_{max} \rceil /2 - s_{max}^2 \lceil \log_2 s_{max} \rceil^2/2 \\ &\quad - g_{max}^2 - g_{max}^2 \lceil \log_2 g_{max} \rceil /2 - g_{max}^2 \lceil \log_2 g_{max} \rceil^2/2 \\ &\quad - s_{max} - s_{max} \lceil \log_2 g_{max} \rceil /2 - s_{max} \lceil \log_2 s_{max} \rceil^2/2) \\ &= O(c_1 \lceil \log_2 n \rceil + c_2), \text{ where } c_1 \text{ and } c_2 \text{ are} \\ &\quad \text{coefficients dependent on } s_{max} \text{ and } g_{max}. \end{aligned} \tag{4}$$

Time complexity of the sequential algorithm proposed by Mitchell [20], [22] is given as follows:

$$T_1(n) = O(s_{max} \times g_{max} \times n + s_{max}^2 \times p + g_{max}^2 \times \bar{p}), \tag{5}$$

where p represents the number of positive instances, $\bar{p}$ represents the number of negative instances, and $p + \bar{p} = n$. Some related remarks about the time complexity analysis are discussed below.

1) If the sequential merging algorithm is used instead of the parallel merging algorithm, then $M_1(n) = O(s_{max}^3 + g_{max}^3 + s_{max} \times g_{max})$, and we have:

$$\begin{aligned} T_n(n) &= T_n/2(n/2) + M_1(n) \\ &= \sum_{k=1}^{\lceil \log_2 n \rceil} M_1(2^k) \\ &= O((s_{max}^3 + g_{max}^3 + s_{max} \times g_{max} \lceil \log_2 n \rceil). \end{aligned} \tag{6}$$

2) If $n \times \max(s_{max}, g_{max})$ processors are available, each merging process can then be done in parallel by at least $\max(s_{max}, g_{max})$ processors. In this case, we have the following:

$$\begin{aligned} &T_{n \cdot \max(s_{max}, g_{max})}(n) \\ &= T_{n \cdot \max(s_{max}, g_{max})/2}(n/2) \\ &\quad + M_{n \cdot \max(s_{max}, g_{max})}(n) \\ &= \sum_{k=1}^{\lceil \log_2 n \rceil} M_{2^k \max(s_{max}, g_{max})}(2^k) \\ &= O((s_{max}^2 + s_{max}^2 \lceil \log_2 s_{max} \rceil \\ &\quad + g_{max}^2 + g_{max}^2 \lceil \log_2 g_{max} \rceil \\ &\quad + s_{max} + s_{max} \lceil \log_2 g_{max} \rceil) \lceil \log_2 n \rceil) \\ &= O(c_1 \lceil \log_2 n \rceil). \end{aligned} \tag{7}$$

3) If $s_{max} + g_{max}$ processors are available for the merging process, PSTEP's 2, 3 and PSTEP's 4, 5 can be simultaneously done. Besides, in a way similar to PSTEP 6, |S| hypotheses of S can also be managed in parallel, with PSTEP 7 no longer being needed. In this case, time complexity for contradictory checking is $O(\max(s_{max}, g_{max}))$ instead of $O(s_{max} + s_{max} \lceil \log_2 g_{max} \rceil)$. Since g_{max} is usually larger than s_{max}, $0(\max(s_{max}, g_{max}))$ is not necessarily smaller than $0(s_{max} + s_{max}) \lceil \log_2 g_{max} \rceil)$.
4) In PSTEP's 2, 3, time complexity is $O(s_{max}^3/q + s_{max}^2 \lceil \log_2 q \rceil)$. For $s_{max}^3/(q/2) + s_{max}^2 \lceil \log_2 (q/2) \rceil) \geq s_{max}^3/q + s_{max}^2 \lceil \log_2 q \rceil$, q must be $\leq s_{max}$. It means that applying more than s_{max} processors cannot reduce execution time anymore. This is the reason why only S_1, instead of both S_1 and S_2, needs to be divided and assigned onto the available processors.
5) Time complexity of the sequential learning algorithm proposed by Mitchell may actually be greater than that listed in (5). This is because to exclude a negative training instance out of a hypothesis is more complex than a specialization process from two chosen hypotheses.

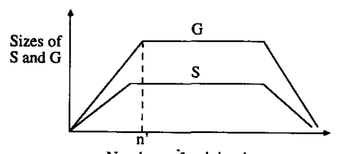

Fig. 6. Typical relation of version space boundary set sizes to numbers of training instances.

6) The maximum numbers $s_{\max}$ and $g_{\max}$ of hypotheses in sets S and G can generally be characterized by the following entities:

$$s_{\max} = Function_s(P, N, O),$$
$$g_{\max} = Function_g(P, N, O),$$

where

$P=$ the particular learning problem considered,
$N=$ the number of training instances, and
$O=$ the order in which training instances are arranged.

Interleaving positive and negative training instances can make illegal hypotheses in S and G removed as early as possible in the contradiction checking phase; it is then the best arrangement of training instances. By interleaving positive and negative training instances, the following equation can then be derived:

$$s_{\max} = Function'_s(P, N),$$
$$g_{\max} = Function'_g(P, N).$$

Mitchell also showed that the sizes of S and G typically behave as shown in Fig. 6 [20]. When n is large ($\geq n'$), then we have the following:

$$s_{\max} = Function''_s(P),$$
$$g_{\max} = Function''_g(P).$$

In the parallel learning algorithm for large n, $s_{\max}$ and $g_{\max}$ then depend only on the learning problem.

Mitchell further pointed out for the feature interval learning problem as in Example 1, S can never contain more than one hypothesis, and the size of G is usually small [20]. Experiments in Section VII show this.

VI. A Bounded Number of Processors

In the algorithm mentioned above, the learning process needs n processors where n (the number of training instances) may be very large. But in real applications, only a limited number of processors are available. It is then necessary to develop a parallel learning algorithm with only a bounded number of processors available.

Assume $r(< n)$ processors are available, where r is a constant. The modified parallel learning algorithm here for a bounded number of processors is divided into two phases. In Phase 1, n given training instances are equally divided and placed on r processors, and the sequential learning algorithm runs on each processor to obtain its own version space. In Phase 2, the r version spaces are merged in a way of binary tree, as mentioned in Section IV. Time complexity of the modified algorithm can be easily analyzed as follows:

$$T_r(n) = O((s_{\max} \times g_{\max} \times n/r + s^2_{\max} \times p/r + g^2_{\max} \times \bar{p}/r) + \sum_{k=1}^{\lceil \log_2 r \rceil} M_{2^k}(2^k)). \tag{8}$$

When the number of available processors increases, the time spent in Phase 1 will decrease; but the time spent in Phase 2 will increase. The time saved in Phase 1 will grow smaller and smaller along with the addition of processors, and, however, does not necessarily make up the additional overhead in Phase 2. It implies in getting to the maximum speedup, the number of processors is not necessarily to be n. The appropriate number r of processors (over which additional processors are useless to increase speedup) can then be decided by the inequation $T_r(n) \leq T_{r/2}(n)$, and the following calculation is derived:

$$r \leq (s_{\max} \times g_{\max} \times n + s^2_{\max} \times p + g^2_{\max} \times \bar{p})/M_r(n). \tag{9}$$

From (9), when n is larger, r is larger, too; speedup is then also larger.

VII. Experiments

In demonstrating effectiveness of the proposed parallel version space learning algorithm, Fisher's Iris Data, containing 150 training instances, is used [6]. Since the data is inconsistent, two training instances are removed out of it to make it consistent, so that the final version space will not be empty. Managing noisy data by version space is beyond discussion here, and can be referenced in [10], [12], [15].

The Iris Problem has been introduced in Example 1. In fact, three species of Iris Flowers to be distinguished exist: setosa, versicolor, and verginica. Fifty training instances exist for each class. When the concept of setosa is learned, training instances belonging to setosa are considered as positive; training instances belonging to the other two classes are considered as negative. Similar management is applied in learning concepts of the other two classes. Execution of the parallel learning algorithm is simulated at the IBM PC/AT (with communication time ignored). By running 100 times, execution time along with different numbers of processors (by the parallel merging algorithm) is shown in Fig. 7. For showing difference between using the parallel merging algorithm and using the sequential merging algorithm, versicolor's data is used as positive instances, with the result being as shown in Fig. 8. The numbers of operations calculated by (8) are also shown in Fig. 9. Sizes of $s_{\max}$ and $g_{\max}$ are listed in Table II.

From these figures, the following points can be observed.

1) From (9), when the number of processors is larger than a certain number r, additional processors will not continue

TABLE II
SIZES OF $s_{\max}$ AND $g_{\max}$

	$s_{\max}$	$g_{\max}$
Setosa	1	8
Viginica	1	9
Versicolor	1	8

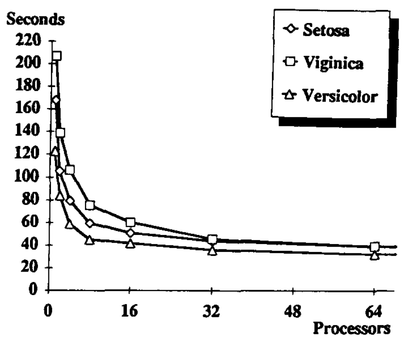

Fig. 7. Execution time for Iris Learning Problem.

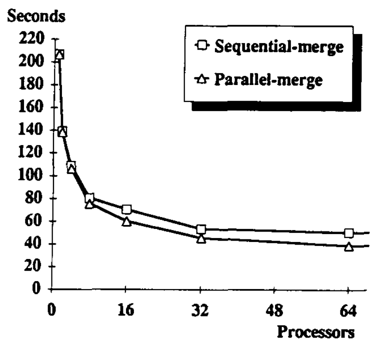

Fig. 8. Execution time of different merging algorithms.

to reduce execution time. Substituting $s_{\max}, g_{\max}, n, p$ and $\bar{p}$ by 1, 8, 150, 50, and 100, respectively, in (9), r is derived to be 16 (power of 2). The best speedup then happens when 16 processors are used. Since (9) is analyzed by the worst case, the shape of curves in Fig. 7 can be thought to be quite consistent with theoretical analysis shown in Fig. 9.

2) From Fig. 8, difference of execution time in using the parallel merging algorithm and using the sequential merging algorithm increases when the number of available processors increases. When the number of processors increases, the merging steps increase and the saved time increases also.
3) The worst-case speedup by theoretical analysis based on (4) and (5) is about 3.92, being close to the empirical speedups, 4.33, 5.27, and 3.84 for these three classes.

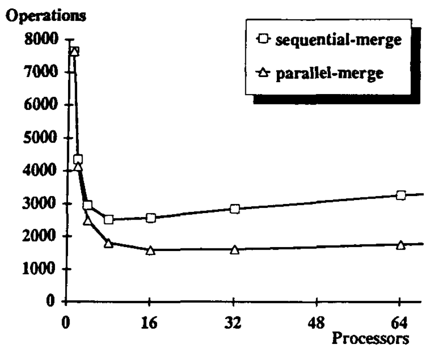

Fig. 9. Numbers of operations calculated by (8).

In comparison with the effect of interleaving positive and negative training instances, positive ones are arranged on one side, and negative ones are arranged on the other side. For this arrangement, $g_{\max}$ can get to 50 and execution time for the sequential learning algorithm can arrive to 69.22 s in average on Class setosa. Note that $g_{\max}$ is only 8 and execution time is only 1.678 s, on the average, for interleaving arrangement. Interleaving positive and negative training instances is then an effective way in reducing sizes of S and G and reducing execution time of the learning process.

At last, the original data is duplicated 10 times in forming a new set that contains 1500 training instances. The parallel learning algorithm and the sequential learning algorithm then run on the set and speedup is found to be 22.73. Theoretic speedup by (4) and (5) is 19.53. Comparing this result with that obtained from only 150 training instances, we can conclude that speedup will increase along with an increase of training instances.

VIII. Conclusion and Prospects for Future Work

We have successfully applied the technique of parallel processing in the field of concept learning for raising the learning speed. We have proposed a parallel version space learning algorithm based upon the principle of divide-and-conquer, and modified this parallel learning algorithm, as a result of practical restrictions, to a bounded number of processors.

By theoretic analysis, we have found, with n processors, that the learning process can be accomplished within $O(\varepsilon_1 \rightarrow c_1 \log_2 n + c_2)$. Only r processors, which can be derived by (9), are actually enough in achieving the maximum speedup. Our parallel learning algorithm has also been applied on the Iris Learning Problem in achieving a result quite consistent with our analysis. Effectiveness of interleaving positive and negative training instances is also verified.

One important point must finally be clarified here. Mitchell mentioned that when the current version space is incompletely learned, an informative new training instance that matches half the hypotheses in the current version space would be selected in the next learning iteration [20]. A shortest number of training instances are then enough to make a version space converge to a single hypothesis, thought of as the final concept. This strategy is good and effective for some learning problems; however, it is not totally suitable for the learning problem defined in Section II. In this paper, the learning problem to be solved is to find hypotheses consistent with given training instances. When the training instances are inconsistent or the description language is insufficient, the version space derived is null. A final version space with only a single hypothesis can always be derived by using only the shortest number of informative new training instances. This version space derived, however, cannot guarantee consistency with all the given training instances.

Besides, it may be not easy to find a training instance that matches half (or nearly half) the hypotheses in a version space. Mitchell proposed an effective heuristic for the feature interval learning problems. In general, however, an analysis for a general learning problem is complex and costly [20]. Furthermore, even the training instance that matches half the hypotheses in the current version space can be derived, it may be out of the given training set. Methods for collecting additional training instances are then required, and the learning job must then be delayed. As a better alternative than that, all the current training instances are processed. If a convergent version space or a null version space is derived, then the learning process finishes; otherwise, the incompletely learned version space can also serve as a better classifier than that by the former alternative. Off-line knowledge acquisition is then performed, if possible, to find informative new training instances for further shrinking the incompletely learned version space.

We must then depend on the characteristics of the application domains to determine an appropriate learning strategy. If the given training instances are assured to be consistent, each informative new training instance is easily calculated from a version space, and if all possible instances exist in the given training set, then sequential learning by processing an informative new training instance at each iteration is good. Otherwise, processing all the given training instances would provide a better result, and parallel processing can then be well applied.

One disadvantage of the version space learning strategy is that it is sensitive to noise. When noise exists in the training set, the version space derived is usually null. Noise, however, exists in nearly all real-world application domains. Developing a new learning strategy or modifying an existing learning strategy for managing noise or uncertainty is then necessary. We have successfully proposed a generalized version space learning strategy [12], [15] to handle training sets with noisy and uncertain instances. We are now trying to use the parallel learning model for the kind of learning problems.

For other models of parallel machine learning, we have applied the principle of task assignment to top-down learning strategies [13]. We have also applied the broadcasting communication model in simultaneously processing training instances in connectionist learning [14]. In the future, we will attempt to develop parallel models for other learning methods and to implement them on various types of parallel architectures.

ACKNOWLEDGMENT

The authors would like to thank C.-T. Yang for his assistance in performing these experiments. We would also like to thank the anonymous referees for their very constructive comments.

REFERENCES

[1] S. G. Akl, *The Design and Analysis of Parallel Algorithms.* Englewood Cliffs, NJ: Prentice-Hall, 1989, pp. 10–17.

[2] B. Arbab and D. Michie, "Generating rules from examples," in *Proc. 1985 Int. Joint Conf. Art. Intell.*, 1985, pp. 631–633.

[3] B.G. Buchanan and T.M. Mitchell, "Model-directed learning of production rules," in D.A. Waterman and F. Hayes-Roth, Eds., *Pattern-Directed Inference Systems.* New York: Academic, 1978, pp. 297–312.

[4] A. Bundy, B. Silver, and D. Plummer, "An analytical comparison of some rule-learning programs," *Art. Intell.*, vol. 27, pp. 137–181, 1985.

[5] T.G. Dietterich and R.S. Michalski, "A comparative review of selected methods for learning from examples," in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Eds., *Machine Learning: An Artificial Intelligence Approach*, vol. 1. Palo Alto, CA: Toiga, 1983, pp. 41–81.

[6] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, pp. 179–188, 1936.

[7] L.M. Fu and B.G. Buchanan, "Learning intermediate concepts in constructing a hierarchical knowledge base," in *Proc. 1985 Int. Joint Conf. Art. Intell.*, 1985, pp. 659–666.

[8] S.I. Gallant, "Automatic generation of expert systems from examples," in *Proc. 2nd Int. Conf. Art. Intell. Applications*, 1985, pp. 313–319.

[9] A. Gupta, "Implementing OPS5 production systems on DADO," in *Proc. 1984 IEEE Int. Conf. Parallel Processing*, 1984, pp. 83–91.

[10] H. Hirsh, "Incremental version-space merging: a general framework for concept learning," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 1989.

[11] T.P. Hong and S.S. Tseng, "A parallel concept learning algorithm based upon version space strategy," in *Proc. 1990 IEEE Int. Phoenix Conf. Comput. Commun.*, 1990, pp. 734–740.

[12] ——, "A generalized learning problem," in *Proc. 1990 Int. Comput. Symp.*, 1990, pp. 517–522.

[13] ——, "Models of parallel learning systems," in *Proc. 1991 IEEE Int. Conf. Distrib. Computing Syst.*, 1991, pp. 125–132.

[14] ——, "Parallel perceptron learning on a single-channel broadcast communication model," *Parallel Computing*, vol. 18, pp. 133–148, 1992.

[15] T.P. Hong, "A study of parallel processing and noise management on machine learning," Ph.D. dissertation, National Chiao Tung Univ., Taiwan, Republic of China, Jan. 1992.

[16] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* New York: Computer Science Press, 1978, pp. 98–140.

[17] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*, vol. 1. Palo Alto, CA: Toiga, 1983.

[18] ——, *Machine Learning: An Artificial Intelligence Approach*, vol. 2. Palo Alto, CA: Toiga, 1984.

[19] T.M. Mitchell, "Version space: a candidate elimination approach to rule learning," in *Proc. 1977 Int. Joint Conf. Art. Intell.*, 1977, pp. 305–310.

[20] ——, "Version spaces: An approach to concept learning," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 1978.

[21] ——, "An analysis of generalization as a search problem," in *Proc. 1979 Int. Joint Conf. Art. Intell.*, 1979, pp. 577–582.

[22] ——, "Generalization as search," *Art. Intell.*, vol. 18, pp. 203–226, 1982.

[23] T.M. Mitchell, P.E. Utgoff, and R. Banerji, "Learning by experimentation: Acquiring and refining problem-solving heuristics," in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Eds., *Machine Learning: An Artificial Intelligence Approach*, vol. 1. Palo Alto, CA: Toiga, 1983, pp. 163–190.

[24] G.D. Plotkin, "A note on inductive generalization," in B. Meltzer and D. Michie, Eds., *Machine Intelligence*, vol. 5. New York: Wiley, 1970, pp. 153–163.

[25] ——, "A further note on inductive generalization," in B. Meltzer and D. Michie, Eds., *Machine Intelligence*, vol. 6. New York: Wiley, 1971, pp. 101–124.

[26] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers.* New York: McGraw-Hill, 1987.

[27] R.C.T. Lee, R.C. Chang, and S.S. Tseng, *Introduction to Design and Analysis of Algorithms.* Englewood Cliffs, NJ: Prentice-Hall, 1991.

[28] M. J. Shaw, "Applying inductive learning to enhance knowledge-based expert systems," *Decision Support Syst.*, vol. 3, pp. 319–332, 1987.
[29] S. J. Stolfo and D. P. Miranker, "DADO: A parallel processor for expert systems," in *Proc. 1984 IEEE Int. Conf. Parallel Processing*, 1984, pp. 74–82.
[30] L. H. Witten and B. A. Macdonald, "Using concept learning for knowledge acquisition," *Int. J. Man-Mach. Studies*, vol. 29, pp. 171–196, 1988.

T.-P. Hong received the B.S. degree in chemical engineering from National Taiwan University in 1985, and the Ph.D. degree in computer science and information engineering from National Chiao Tung University in 1992.

Since 1987, he has been with the Laboratory of Knowledge Engineering, National Chiao Tung University, where he has been involved in applying techniques of parallel processing to artificial intelligence. He is currently an Associate Professor at the Department of Computer Science at Chung-Hua Polytechnic Institute in Taiwan, Republic of China. His current research interests include parallel processing, machine learning, neural networks, fuzzy set, and expert systems.

S.-S. Tseng (M'88) received the B.S., M.S., and Ph.D. degrees in computer engineering from National Chiao Tung University in 1979, 1981, and 1984, respectively.

Since August 1983, he has been on the faculty of the Department of Computer and Information Science, National Chiao Tung University, and is currently a Professor there. From 1988 to 1991, he was the Director of the Computer Center at National Chiao Tung University, Taiwan, Republic of China. From 1991 to 1992, he acted as the Chairman of the Department of Computer and Information Science. Now he is the Director of the Computer Center at the Ministry of Education, Republic of China. His current research interests include parallel compiler, language processor design, computer algorithms, parallel processing, and expert systems.

Dr. Tseng is an Associate Editor of *Information and Education*, and a member of the International Advisory Board of CISNA COMPUTER FORUM. He was elected an Outstanding Talent of Information Science of the Republic of China in 1989. He was also awarded the Outstanding Youth Honor of the Republic of China in 1992.