Incremental Clustering of Attributed Graphs

Dong Su Seong, *Member, IEEE*, Ho Sung Kim, *Member, IEEE*, and Kyu Ho Park, *Member, IEEE*

Abstract—**An incremental clustering system based on a new criterion function to group patterns represented by attributed graphs is presented. The system takes a succession of attributed graphs and builds up a concept hierarchy that summarizes and organizes input instances incrementally. For this purpose, it is proposed that a new criterion function based on entropy minimization, and present the incremental clustering algorithm with the criterion function. For the attributed graph as an input instance, the clustering algorithm incrementally obtains a concept hierarchy in a top-down manner using the hill climbing strategy. It is shown that the execution of the incremental clustering algorithm and classification algorithm can be interleaved since the concept hierarchy is constructed incrementally. Finally, the proposed method is applied to the clustering of simple examples to show its capability.**

I. Introduction

AN ATTRIBUTED graph was defined to represent both structural and semantic information contained in structural objects [1]. In the attributed graph, the underlying graph represents the global structure of the given object, and the semantic information in the vertices and edges specify the local properties. It can be employed for the structural description that describes a certain object in terms of its parts and their mutual relations. It can be applied to many areas such as syntactic pattern recognition, scene analysis, chemical structure descriptions, relational databases, and so on [1], [2], [7].

There are many interesting problems in handling attributed graphs, such as attributed graph matching and clustering of attributed graphs. In particular, clustering is one of the important areas in machine learning and pattern recognition. It is concerned with organizing objects in a form of concept hierarchy, which will eventually ease the problem of knowledge acquisition for a knowledge based or expert system in classifying future experiences.

Fisher [11] described incremental clustering with category utility as its criterion function. However, they studied the attributed graph with no edge attribute. Wong [6] treated the clustering of attributed graphs by an agglomerative method with a distance measure between random graphs, but this yields no incremental capability. The clustering problem can be characterized as efforts to minimize the entropy [14], and Wallance [15] used the criterion function based on entropy minimization to find natural clusters, but this method also did not have incremental capabilities and did not handle attributed graphs as instances. Our motivation is derived from a need to hierarchically group the structural objects represented in attributed graphs and to reflect more accurately a real-world situation in which learning should be an ongoing process [17].

In this paper, we propose a new criterion function based on the entropy minimization for incremental clustering of attributed graphs. As an application example, the incremental clustering system is implemented to group the patterns represented by attributed graphs. The system takes a succession of attributed graphs and builds up a concept hierarchy that summarizes and organizes the input instances. The system carries out the hill climbing search through the space of possible concept hierarchies using the criterion function. The concept hierarchy is incrementally constructed. This means that the clustering, called a learning, and classification can be interleaved, that is, the concept hierarchy for the classification needs not wait for all the input instances, but is available after processing each instance.

The remainder of this paper is organized as follows. Section II defines the new criterion function based on entropy minimization, which is used to evaluate the various clusterings in the incremental clustering algorithm. Section III describes the incremental clustering algorithm. Application example of the proposed method is presented in Section IV. Finally, in Section V, we present our conclusions.

II. Criterion Function Based on Entropy Minimization

The clustering problem can be regarded as an energy optimization problem to which any standard minimization technique applies. Entropy minimization is one of possible techniques.

In this section, we propose a new criterion function based on entropy minimization, which is required to evaluate the various clusterings in the incremental clustering algorithm. This section is organized as follows. We first describe definitions of attributed and random graphs. Second, definitions of attributed random graphs which are used to represent a concept, called class, are introduced, and equations for entropy calculation of attributed random graphs are derived. Finally, a new criterion function based

Manuscript received July 28, 1991; revised August 2, 1992 and February 3, 1993.

D. S. Seong and K. H. Park are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Taejon 305-701, Korea.

H. S. Kim is with the Department of Computer Science, Sungshin Womens University, Seoul 136-701, Korea.

IEEE Log Number 9209680.

on entropy minimization for the incremental clusterings is introduced.

A. Definition of Attributed and Random Graphs

The basic concept and definitions given in this subsection mainly come from several other works [1], [5]–[7]. They are rewritten here for the definition of the attributed random graph in the following subsection.

Definition 1: An attributed vertex v and edge e are associated with an attribute set in which each element is an attribute pair consisting of an attribute name and its attribute value X. The value X can be a nominal value or a numeric value. For example, an attribute vertex v can be used to describe a component of a structural object such as

$$v = [(\text{shape rectangle})(\text{compactness } 0.9)(\text{area } 0.8)].$$

Definition 2: An attributed graph is defined to be a graph $G_a = (V_a, E_a)$, where $V_a = \{v_1, \cdots, v_p, \cdots, v_q, \cdots, v_n\}$ is a set of attributed vertices and $E_a = \{\cdots, e_{pq}, \cdots\}$ is a set of attributed edges, and the edge e_{pq} connects vertices v_p and v_q with an attributed relation. As an example, an object decomposed into two components is illustrated in Fig. 1.

Definition 3: A random vertex α and edge β consist of a pair of null probability and nonnull probability $\langle P_\alpha \quad Q_\alpha \rangle$ and $\langle P_\beta \quad Q_\beta \rangle$, respectively. For example, if random vertex α is represented such as $\alpha = \langle 0.2 \quad 0.8 \rangle$, the probability for the outcome of random vertex α to be null is 0.2.

In fact, we only deposited P_α, P_β for each random vertex α, random edge β because of $Q_\alpha = 1 - P_\alpha$, $Q_\beta = 1 - P_\beta$.

Definition 4: A random graph is defined to be a graph $R = (A, B)$, where $A = \{\alpha_1, \cdots, \alpha_p, \cdots, \alpha_q, \cdots, \alpha_m\}$ is a set of random vertices and $B = \{\cdots, \beta_{pq}, \cdots\}$ is a set of random edges, and the random edge β_{pq} connects random vertices α_p and α_q with a random relation. For example, Fig. 2(a) represents a random graph R consisting of random vertices $(\alpha_1, \alpha_2, \alpha_3)$ and random edges (β_{12}, β_{13}). Fig. 2(b) represents the null and nonnull probability of attributed random vertices and edges.

Definition 5: In a random graph, if either endpoint of a random edge has a null outcome, the outcome of the random edge would be null with probability one.

$$\text{Prob}\{\beta = \phi \mid \sigma(\beta) = \phi \text{ or } \tau(\beta) = \phi\} = 1 \qquad (1)$$

where $\sigma(\beta)$ and $\tau(\beta)$ are the initial and terminal random endpoints of a random edge β, respectively.

Definition 6: The probability space of a random graph R consists of outcome graphs G with a graph monomorphism M: $G \rightarrow R$. The M can be denoted by two sets of mapping (μ, γ), μ: $v \rightarrow \alpha$ and γ: $e \rightarrow \beta$. The $\mu(v)$ and $\gamma(e)$ stand for the random vertex α and edge β, and $\mu^{-1}(\alpha) = v$, $\gamma^{-1}(\beta) = e$. There is a probability $P(G, M)$ that satisfies

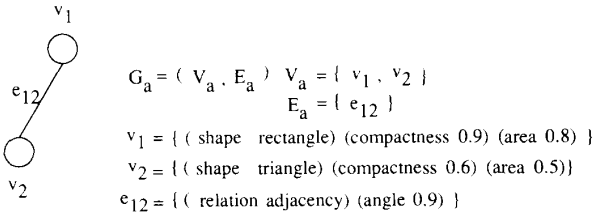
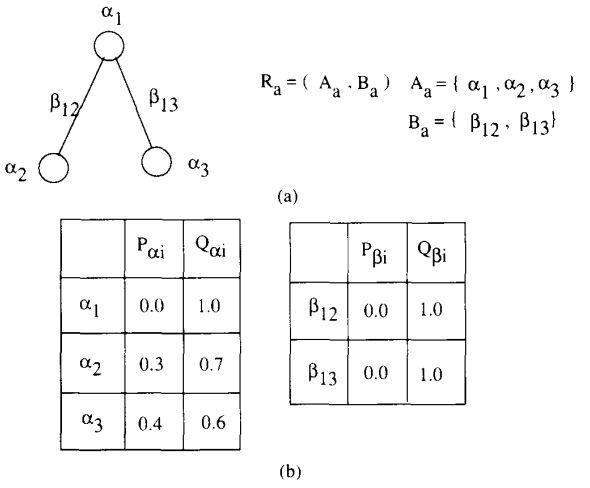


Fig. 1. An example of attributed graph.



	$P_{\alpha i}$	$Q_{\alpha i}$
α_1	0.0	1.0
α_2	0.3	0.7
α_3	0.4	0.6

	$P_{\beta i}$	$Q_{\beta i}$
β_{12}	0.0	1.0
β_{13}	0.0	1.0

Fig. 2. (a) The example of random graph. (b) The null and nonnull probability of random vertices and edges.

$$P(G, M) \geq 0 \qquad \text{for all } G \in \Gamma \qquad (2)$$
$$\sum_{G \in \Gamma} P(G, M) = 1$$

where Γ is the range of R.

B. Definitions and Entropy Calculation of Attributed Random Graph

In this subsection, the definitions of an attributed random graph are introduced by including multiple attributes as well as numeric attributes instead of a single nominal attribute [5], [6] in random vertices and edges. The equations for the entropy calculation of the attributed random graph are then presented. From the equations, the criterion function for incremental clustering will be defined in the next subsection.

Definition 7: An attributed random pair is an ordered pair that consists of the attribute name and the random variable x. In case of a nominal attribute, the x is represented with a list on which each element consists of a value and the probability of this value. For example, an attributed random pair can be used to describe the shape of an object made with many instances such as

$$\text{(shape } x\text{), where the random variable } x = [(\text{triangle } 2/3)(\text{rectangle } 1/3)].$$

This means that the probability for the shape of the object to be a triangle and a rectangle is 2/3 and 1/3, respectively.

In the case of a numeric attribute, the representation of random variable x is classified into two phases. If x is made by small instances, it is treated the same as the nominal attribute. But with sufficient instances, we assume that the distribution of the variable follows a normal distribution since it can be generally accepted for the distribution of a numeric attribute [11], therefore the random variable x is represented with its mean and the standard deviation. For example, an attributed random pair can be used to describe the compactness of an object with mean 0.8 and standard deviation 0.4 such as

(compactness x), where the random variable x

$= (0.8 \quad 0.4)$.

In fact, our system does not directly store the mean μ_x and standard deviation σ_x for the numeric attribute x, instead, it stores a frequency f, a sum of value ΣX, a sum of square value ΣX^2 from which the mean and standard deviation is computed incrementally [11] by (3).

$$\sigma_x = \sqrt{\frac{\sum_{X \in x} X^2}{f} - \mu_x^2}$$

$$\mu_x = \frac{\sum_{X \in x} X}{f}. \tag{3}$$

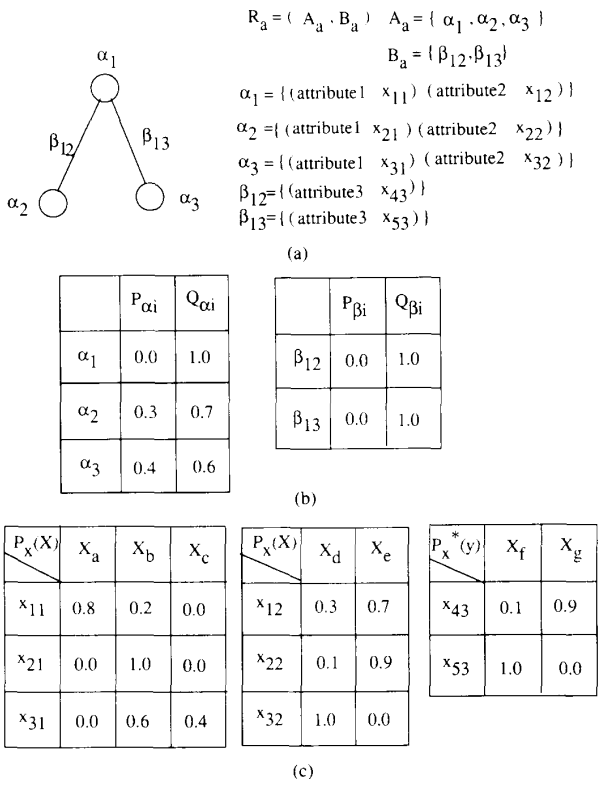
Definition 8: An attributed random vertex α and edge β consist of a set of attributed random pairs.

Definition 9: An attributed random graph is defined to be a random graph $R_a = (A_a, B_a)$, where $A_a = \{\alpha_1, \cdots, \alpha_p, \cdots, \alpha_q, \cdots, \alpha_m\}$ is a set of attributed random vertices and $B_a = \{\cdots, \beta_{pq}, \cdots\}$ is a set of attributed random edges. For example, Fig. 3(a) represents an attributed random graph R consisting of attributed random vertices (α_1, α_2, α_3) and attributed random edges (β_{12}, β_{13}). Fig. 3(b) represents the null and nonnull probability of attributed random vertices and edges. Fig. 3(c) shows the probability distribution of random variables in attributed random vertices and edges, in which we assume that the domains of attribute1, attribute2, and attribute3 are $\{X_a, X_b, X_c\}$, $\{X_d, X_e\}$, and $\{X_f, X_g\}$, respectively.

Definition 10: The outcome of the attributed random graph $R_a = (A_a, B_a)$ is an attributed graph $G_a = (V_a, E_a)$. Fig. 4 shows an outcome G_a of the attributed random graph R_a of Fig. 3 with monomorphism M.

The probability space of joint distribution consisting of all random vertices and random edges can be partitioned into infeasible and feasible subsets. The infeasible subsets consist of all joint outcomes that contain at least one edge whose either endpoint is null. Therefore, an outcome of infeasible subsets is not the output of an attributed random graph according to (1). The feasible subsets are the complement of the infeasible subsets.

Therefore, the probability of an outcome graph can be



	$P_{\alpha i}$	$Q_{\alpha i}$
α_1	0.0	1.0
α_2	0.3	0.7
α_3	0.4	0.6

	$P_{\beta i}$	$Q_{\beta i}$
β_{12}	0.0	1.0
β_{13}	0.0	1.0

$P_x(X)$	X_a	X_b	X_c
x_{11}	0.8	0.2	0.0
x_{21}	0.0	1.0	0.0
x_{31}	0.0	0.6	0.4

$P_x(X)$	X_d	X_e
x_{12}	0.3	0.7
x_{22}	0.1	0.9
x_{32}	1.0	0.0

$P_x^*(y)$	X_f	X_g
x_{43}	0.1	0.9
x_{53}	1.0	0.0

Fig. 3. (a) The example of attributed random graph. (b) The null and nonnull probability of random vertices and edges. (c) The probability distributions of attribute value in random vertices and edges.

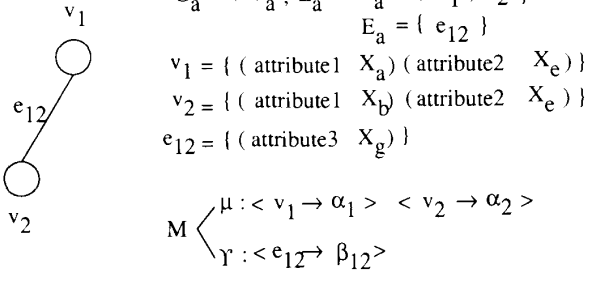


Fig. 4. An outcome of the attributed random graph.

defined to be equal to the probability of the corresponding joint outcome in the probability space that the probability of infeasible subsets is 0.

$$P(G_a, M) = \text{Prob}\ \{(\alpha = \mu^{-1}(\alpha), \forall \alpha \in A_a), (\beta = \gamma^{-1}(\beta), \forall \beta \in B_a)\}. \tag{4}$$

It is necessary to reduce the dimensionality of the joint probability space for the estimation of the probabilities of outcome graphs in real applications. Thus, the following assumptions [5] are made and these are generally reasonable in real situations.

1) The random vertices α, $\alpha \in A_a$ are mutually independent.

2) The random vertices α, $\alpha \in A_a$ are independent of any random edge.

3) The random edge β, $\beta \in B_a$ are mutually independent.

4) The random edge β, $\beta \in B_a$ is independent of any random vertex other than its endpoint $\sigma(\beta)$, and $\tau(\beta)$.

5) Attributed random pairs in a random vertex or edge are mutually independent.

6) If the outcomes of endpoints of a random edge are both nonnull, then the probability of any outcome of the random edge is the same regardless of the actual outcomes of the endpoints.

From the above assumptions, the probability of an outcome graph $P(G_a, M)$ can be expressed by

$$P(G_a, M) = \prod_{\alpha \in A_a} \text{Prob}\{\alpha = \mu^{-1}(\alpha)\} \prod_{\beta \in B_a'} \cdot \text{Prob}\{\beta = \gamma^{-1}(\beta) \mid \sigma(\beta) \neq \phi, \tau(\beta) \neq \phi\} \tag{5}$$

where $B_a' = \{\beta \mid \mu^{-1}[\sigma(\beta)] \neq \phi, \mu^{-1}[\tau(\beta)] \neq \phi \text{ for } \beta \in B_a\}$, the B_a' is the set of random edges in which the outcomes of endpoints of each random edge β are nonnull in the outcome graph G_a.

To rewrite above equation in a manageable form, the following notations [5] are used.

$$P_\alpha = \text{Prob}\{\alpha = \phi\} \tag{6}$$

$$Q_\alpha = 1 - P_\alpha \tag{7}$$

$$P_\beta = \text{Prob}\{\beta = \phi \mid \sigma(\beta) \neq \phi, \tau(\beta) \neq \phi\} \tag{8}$$

$$Q_\beta = 1 - P_\beta \tag{9}$$

$$C_\beta = \text{Prob}\{\sigma(\beta) \neq \phi, \tau(\beta) \neq \phi\} \tag{10}$$

$$P_\alpha(v) = \text{Prob}\{\alpha = v\} = Q_\alpha \prod_{x_i \in \alpha} P_{x_i}(X_i) \tag{11}$$

$$P_\beta(e) = \text{Prob}\{\beta = e \mid \sigma(\beta) \neq \phi, \tau(\beta) \neq \phi\} \tag{12}$$

$$= Q_\beta \prod_{x_i \in \beta} P_{x_i}(X_i \mid \sigma(\beta) \neq \phi, \tau(\beta) \neq \phi) \tag{13}$$

$$= Q_\beta \prod_{x_i \in \beta} P^*_{x_i}(X_i) \tag{14}$$

where $P_\alpha(v)$ is the probability for the outcome of random vertex α to become the attributed vertex v. $P_{x_i}(X_i)$ is the probability for the outcome of random variable x_i is a random vertex to be attribute value X_i when a corresponding random vertex is a nonnull outcome, x_i is a random variable of the ith attribute pair in an attributed random vertex or edge, and ϕ is null outcome. $P^*_{x_i}(X_i)$ is the probability for the outcome of random variable x_i in a random edge to be attribute value X_i when the outcomes of endpoint of a random edge and itself are both nonnull. From (6)–(14), the probability of an outcome graph can be expressed as follows:

$$P(G_a, M) = \prod_{\alpha \in A_a} P_\alpha[\mu^{-1}(\alpha)] \prod_{\beta \in B_a'} P_\beta[\gamma^{-1}(\beta)]. \tag{15}$$

The meaning of this formula is that the probability of outcome graph G_a is computed by the probability for joint outcome of random vertices in A_α and edges in B_a' to become outcome graph G_a. For example, using (15), we obtain the probability of the outcome graph G_a in Fig. 4 by

$$\begin{aligned} P(G_a, M) &= P_{\alpha_1}(v_1) \times P_{\alpha_2}(v_2) \times P_{\alpha_3}(\phi) \times P_{\beta_{12}}(e_{12}) \\ &= Q_{\alpha_1} P_{x_{11}}(X_a) P_{x_{12}}(X_e) Q_{\alpha_2} P_{x_{21}}(X_b) \\ &\quad \cdot P_{x_{22}}(X_e) P_{\alpha_3} Q_{\beta_{12}} P^*_{x_{43}}(X_g) \\ &= 0.127. \end{aligned}$$

To reflect the variability of an attributed random graph, an entropy measure [5] is used. Among the various properties [16] of entropy measure, we introduce the properties used in this paper.

The set of complete finite ($n$ary) probability distribution Γ_n can be expressed by (16) for any natural number n as

$$\Gamma_n = \{(p_1, p_2, \cdots, p_n): \sum_{i=1}^{n} p_i = 1; \; p_i \geq 0; \; n = 1, 2, 3, \cdots\}. \tag{16}$$

We abbreviate the n-tuple as a vector

$$\vec{P} = (p_1, p_2, \cdots, p_n) \tag{17}$$

and the entropy of $\vec{P}$ is defined as follows:

$$H_n(\vec{P}) = -\sum_{k=1}^{n} p_k \log p_k. \tag{18}$$

Property 1—Additivity:

$$H_{mn}(p_1 q_1, \cdots, p_n q_n, p_2 q_1, \cdots, p_m q_n) = H_m(\vec{P}) + H_n(\vec{Q}) \quad \text{for } \vec{P} \in \Gamma_m \text{ and } \vec{Q} \in \Gamma_n.$$

Property 2—Strong additivity:

$$H_{mn}(p_1 q_{11}, \cdots, p_1 q_{1n}, p_2 q_{21}, \cdots, p_m q_{mn}) = H_m(\vec{P}) + \sum_{j=1}^{m} p_j H_n(\vec{Q}_j)$$

for $\vec{P} \in \Gamma_m$ and $\vec{Q}_j \in \Gamma_n$.

Property 3—Concavity:

$$\sum_{i=1}^{m} a_i H_n(\vec{P}_i) \leq H_n\left(\sum_{i=1}^{m} a_i \vec{P}_i\right)$$

for all $\vec{P}_i \in \Gamma_n$ and $\vec{A} \in \Gamma_m$.

From the definition of entropy, the entropy of an attributed random graph is defined by

$$H(R_a) = -\sum_{(G_a, M) \in \Gamma} P(G_a, M) \log P(G_a, M) \tag{19}$$

where Γ is the range of R_a. If there is a significant similarity among a large number of outcome graphs, then the value of $H(R_a)$ is low; otherwise it is high.

From (15) and (19) with property 1 and property 2, we obtain

$$H(R_a) = \sum_{\alpha \in A_a} H_\alpha + \sum_{\beta \in B_a} H_\beta. \tag{20}$$

The entropy of the attributed random vertex α in equation (20) is given by

$$H_\alpha = -\sum_{v} P_\alpha(v) \log P_\alpha(v) \tag{21}$$

where the sum is over all outcomes of random vertex α including the null outcome. If we apply (11) to (21) with property 1 and property 2, we obtain

$$H_\alpha = -Q_\alpha \sum_{x_i \in \alpha} \sum_{X_i \in x_i} P_{x_i}(X_i) \log P_{x_i}(X_i) - P_\alpha \log P_\alpha - Q_\alpha \log Q_\alpha \tag{22}$$

where $P_{x_i}(X_i)$ is the probability for the outcome of random variable x_i to be attribute value X_i when a corresponding random vertex or edge is not a null outcome, x_i is a random variable of the ith attribute pair in an attributed random vertex or edge.

The conditional entropy of the random edge β in (20) is given by

$$H_\beta = -C_\beta \sum_{e} P_\beta(e) \log P_\beta(e). \tag{23}$$

This equation is computed by the (14) with property 1 and property 2 such as

$$H_\beta = -C_\beta Q_\beta \sum_{x_i \in \beta} \sum_{X_i \in x_i} P^*_{x_i}(X_i) \log P^*_{x_i}(X_i) - C_\beta P_\beta \log P_\beta - C_\beta Q_\beta \log Q_\beta. \tag{24}$$

If random variable x_i represents the numeric attribute and it is made by sufficient instances, we assumed the distribution of value follows a normal curve in Definition 7. Therefore, the innermost summation of above equations can be replaced with the following equation.

$$\sum_{X_i \in x_i} P_{x_i}(X_i) \log P_{x_i}(X_i) \tag{25}$$

$$\approx \int_{-\infty}^{\infty} \left\{ \frac{1}{\sqrt{2\pi}\,\sigma_{x_i}} \exp\left(-\left(\frac{x_i - \mu_{x_i}}{\sigma_{x_i}} \right)^2 \right) \Big/ 2 \right\} \cdot \log \left\{ \frac{1}{\sqrt{2\pi}\,\sigma_{x_i}} \exp\left(-\left(\frac{x_i - \mu_{x_i}}{\sigma_{x_i}} \right)^2 \right) \Big/ 2 \right\} dx_i$$

$$= [1 + \log(2\pi) + 2 \log(\sigma_{x_i})]/2. \tag{26}$$

where σ_{x_i} is the standard deviation for random variable x_i. As an example of an entropy calculation, using (20), (22), and (24), we can compute the entropy of the attributed random graph R_a in Fig. 3 by

$$H(R) = H_{\alpha_1} + H_{\alpha_2} + H_{\alpha_3} + H_{\beta_{12}} + H_{\beta_{13}} = 1.111 + 0.838 + 1.077 + 0.228 + 0 = 3.254.$$

Where the entropies of the random vertex α_1 and edge β_{12} are computed by

$$\begin{aligned} H_{\alpha_1} &= -Q_{\alpha_1}\{P_{x_{11}}(X_a) \log P_{x_{11}}(X_a) + P_{x_{11}}(X_b) \log P_{x_{11}}(X_b) + P_{x_{12}}(X_d) \log P_{x_{12}}(X_d) + P_{x_{12}}(X_e) \log P_{x_{12}}(X_e)\} \\ &\quad -P_{\alpha_1} \log P_{\alpha_1} - Q_{\alpha_1} \log Q_{\alpha_1} \\ &= 1.111. \\ H_{\beta_{12}} &= -C_{\beta_{12}} Q_{\beta_{12}}\{P^*_{x_{43}}(X_f) \log P^*_{x_{43}}(X_f) + P^*_{x_{43}}(X_g) \log P^*_{x_{43}}(X_g)\} \\ &\quad -C_{\beta_{12}} P_{\beta_{12}} \log P_{\beta_{12}} - C_{\beta_{12}} Q_{\beta_{12}} \log Q_{\beta_{12}} \\ &= 0.228. \end{aligned}$$

C. Criterion Function for Incremental Hierarchical Clustering

The clustering problem may be regarded as an energy optimization problem [9] and the entropy can be selected as criterion function for clustering because of the following three facts. First, the value of $H(R)$ is low if there is a significant similarity among a large number of outcomes of a random graph R [5]. Second, the entropy minimization lies in its equivalence to other intuitively plausible minimization criteria such as maximum likelihood and minimum variance [9]. Third, the clustering problem can be considered to minimize the entropy [14].

If we have input instances that are represented by attributed graphs and we want to partition the instances into N disjoint classes, $C_1, C_2, \cdots, C_N$, where each class is represented by attributed random graph, the criterion function based on entropy minimization can be defined as follows.

$$CF_1 = \sum_{k=1}^{N} P(C_k) H[R(C_k)] \tag{28}$$

where $P(C_k) = f_k/f$ is the relative frequency of class C_k about all instances, f_k is the frequency of class C_k, and f is the total frequency. $R(C_k)$ is the attributed random graph of class C_k and $H[R(C_k)]$ is the entropy value of $R(C_k)$.

Property 3 shows that the entropy of the combined probability distribution is never smaller than the weighted average of the entropies of the probability distributions. Thus the entropy value $H(R)$ of the random graph made up by overall instances is greater than the weighted summation of entropy value of each class.

We can introduce another criterion function based on the entropy minimization from above discussion.

$$CF_2 = H(R) - \sum_{k=1}^{N} P(C_k) H[R(C_k)]. \tag{29}$$

This is similar to (28) except for a maximization of criterion function whose range is from 0 to $H(R)$. The original minimization problem is now converted to maximi-

zation one with the principles of duality in optimization theory.

If we want to partition the input instances into a variable number of classes, this criterion function must be modified because the extrema exists when the number of classes is equal to the number of instances. When each class is a singleton, CF_2 reduces to the maximum value. Therefore, the criterion function for a variable number of classes may be defined as follows:

$$CF_3 = \left\{ H(R) - \sum_{k=1}^{N} P(C_k) H[R(C_k)] \right\} \Big/ N \quad (30)$$

where N is the number of classes, which varies according to the number of classes. This division prevents the number of classes in the clustering from being equal to the number of instances, and allows for comparison of the various possible clusterings of different number of classes.

The output of our algorithm is a concept hierarchy in which the root node represents the set of all input instances, the terminal nodes represent the individual instance, and the internal nodes stand for sets of instances attached to that node. A concept C, called a class, in the concept hierarchy is represented with a data structure consisting of the attributed random graph $R(C)$ and the list of pointers to each child concept $L(C) = \{C_1, C_2, \cdots, C_i, \cdots, C_n\}$, where C_i is the child concept, called subclass, of concept C.

If we have all instances at once and it is assumed that there are no identical instances, the concept hierarchy can be obtained by recursively partitioning each class into disjoint subclasses with the criterion function until each class consists of one instance. An incremental system not only accepts one instance at a time but also does not allow it to retain all previously encountered instances for incorporating the new instance into the existing concept hierarchy by its definition [11]. Therefore, the incremental clustering leads naturally to the integration of learning with classification. In fact, most existing incremental clustering algorithms [10]–[13] acquire a concept hierarchy in a top-down manner with the hill climbing strategy. That is, the input instance is classified through the concept hierarchy using a criterion function and the system reconstructs the concept hierarchy in its passage.

Our system is a kind of incremental system and one of its characteristics is to take the attributed graphs as input instances. Therefore, we use the same control strategy of the general incremental system and adopt the (30) as the criterion function to evaluate the various clusterings made by reconstructing the partial concept hierarchy on path through the concept hierarchy searching.

We formally define the criterion function for evaluating the clustering to have a class C as parent node as follows.

$$CF(C) = \left\{ H[R(C)] - \sum_{k=1}^{N_C} P(C_k) H[R(C_k)] \right\} \Big/ N_C \quad (31)$$

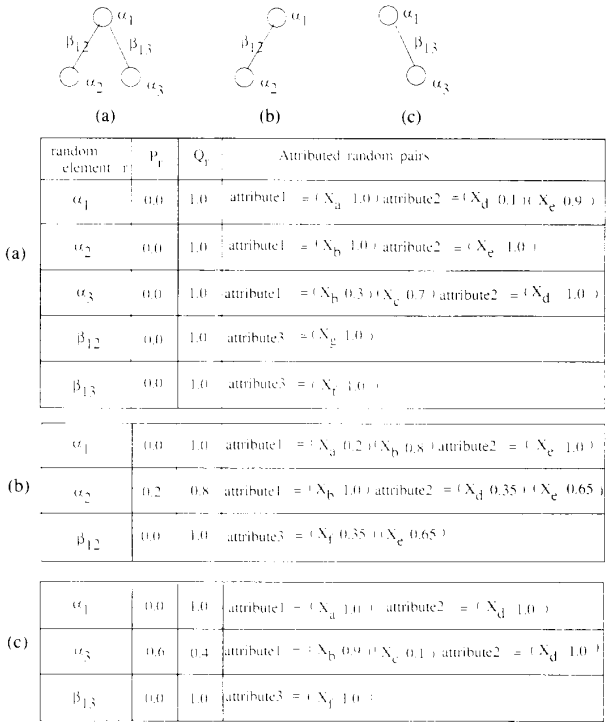


Fig. 5. (a) Attributed random graph $R(C_1)$. (b) Attributed random graph $R(C_2)$. (c) Attributed random graph $R(C_3)$.

where $P(C_k) = f_k/f$ is the relative frequency or a prior probability of subclass C_k in the class C, and f_k, f are the frequencies of C_k and C, respectively. $H[R(C_k)]$ is the entropy of C_k and N_C is the number of subclasses of class C.

We use the (31) as the criterion function of our clustering algorithm, which favors clusterings that minimizes the total weighted entropy of subclasses. It maximizes the difference of entropy between class C and subclasses $L(C)$.

To show an example where a clustering is evaluated by the criterion function, let C be a class and C_1, C_2, and C_3 be the subclasses of class C. We assume that the class C is represented by the random graph $R(C)$ in Fig. 3 and its subclasses C_1, C_2, and C_3 are represented by the random graphs $R(C_1)$, $R(C_2)$, and $R(C_3)$, respectively in Fig. 5(a), (b), and (c). Furthermore we assume that the frequency or the number of instances in C_1, C_2, and C_3 are 50,25, and 25, respectively. Then the relative frequencies $P(C_1)$, $P(C_2)$, and $P(C_3)$ are 1/2,1/4, and 1/4, respectively. The entropy of random graph R_1, R_2, R_3 are calculated to be $H[R(C_1)] = 0.939$, $H[R(C_2)] = 2.036$, $H[R(C_3)] = 0.803$. Using (31), the criterion value of this clustering is computed by

$$\begin{aligned} CF(C) &= \left\{ H[R(C)] - \sum_{k=1}^{3} P(C_k) H[R(C_k)] \right\} \Big/ 3 \\ &= \{3.254 - (0.939/2 + 2.036/4 \\ &\quad + 0.803/4)\}/3 = 0.692. \end{aligned}$$

III. Incremental Clustering Algorithm

In this section, the incremental clustering algorithm of attributed graphs is shown. The goal is to take a succession of attributed graphs and to build up a concept hierarchy that summarizes these input instances.

With the attributed graph as an input instance, the clustering algorithm incrementally acquires a concept hierarchy in a top-down manner with the hill climbing strategy. The algorithm applies one of the five operators: searching, creation, merging, splitting, and termination operators, through states resulting from four tests: searching, creation, merging, and splitting tests, using the criterion function CF defined in (31). It iterates until there is not more progress. The algorithm incrementally incorporates an input instance into the concept hierarchy by descending, updating, and partially reconstructing the concept hierarchy along its passage with proper operators. The clustering algorithm in our system uses similar tests and operators, and the same control strategy as in the previous incremental algorithm [10], [11], but differs mainly in the representation of concepts and the criterion function, because input instances are represented by attributed graphs. Also, in addition, the merging operator is extended from the previous one [10], [11] by the merging of multiple concepts instead of the merging of two concepts.

This section is organized as follows. The first subsection describes the synthesis of two attributed random graphs which is used in the clustering algorithm. In the second subsection, four tests are precisely introduced, which are used for guiding the search through the space for possible concept hierarchies. A proper operator is selected by the results of the tests. The third subsection describes five operators that are used to construct a concept hierarchy incrementally. Finally, in the last subsection, we show that the incremental clustering and classification algorithms can be interleaved by the fact that the concept hierarchy is constructed incrementally.

A. Synthesis of Two Attributed Random Graphs

We use the criterion of minimum entropy for the synthesis of two attributed random graphs. Assuming that the given random graphs R_1 and R_2 retain considerable similarities, we use the entropy as an objective function to find the optimal monomorphism M between R_1 and R_2, the same as in [5] [6]:

$$\min_{M} \{H(R) \mid R = \text{Synthesis } (R_1, R_2) \text{ with } M,\ M \in \Psi\} \tag{32}$$

where Ψ is the set of possible monomorphism between R_1 and R_2. This can be solved by using the graph monomorphism techniques [1], [2] and we use a uniform-cost algorithm with entropy as the cost function. If there are several monomorphisms with the same entropy value, we select the monomorphism to have the minimum order.

Now, two random graphs R_1 and R_2 are synthesized by merging the corresponding random vertices and edges with monomorphism M and incorporating all the unmatched vertices edges in R_1 and R_2.

$$R = \text{Synthesis } (R_1, R_2). \tag{33}$$

B. Tests

Like other incremental algorithms [10]–[12], our clustering algorithm uses four tests to guide the search. These tests do not modify the concept hierarchy that was built from the previous instances but are used to guide the search through the space of possible concept hierarchies for the incorporation of a new instance.

- *Searching Test:* In order to determine which class below concept C best hosts an input instance I during the concept hierarchy searching, the algorithm places the input instance in each class C_i. Each of these constitutes an alternative clustering with common parent node C, and is evaluated using the criterion function CF defined by (31). It then selects the best such clustering that has the maximum value among these clusterings. Let this value be $S_{searching}$ and the procedure is introduced in Table I-A. In this procedure, the function $Copy(A, B)$ copies the data structure, attributed random graph $R(A)$, and the list of subconcepts $L(A)$ of concept A to concept B.
- *Creation Test:* Instead of searching below concept C in the concept hierarchy, a new class below concept C is created. The clustering modified by the addition of a new class that contains only the input instance is evaluated using the criterion function. Let this value be $S_{creation}$ and the procedure is introduced in Table I-B.
- *Merging Test:* After the algorithm evaluates each clustering C modified by the incorporation of an input instance in each class C_i, it sorts the classes with this evaluated scores. Next, for selecting the clustering of a high evaluation score through the merging, the concepts in the sorted list are merged in order until high score is found. For instance, if k classes are merged, the original n classes are transformed into $n - k + 1$ classes. The procedure is introduced in Table I-C. In this procedure, the function $First\_concept(S)$ pops the first concept from the concept list S, and the function $Sorting(C, I)$ returns the children list of concept C sorted through each score that is acquired by evaluating each clustering C after the incorporation of instance I to each child C_i. In the merging test, we consider the merging of the multiple concepts instead of merging two concepts that best classify the new instance in the previous algorithm [10], [11]. As a result, it would be costly but effective in initial skewed instances.
- *Splitting Test:* The splitting test is considered only for the best class C, that is selected in the searching test. The algorithm splits the best class by replacing this class with child classes of this class. As a result,

TABLE I
A. SEARCHING_TEST

```
Procedure Searching_test(C, I)
1.  begin
2.    Max_score ∅, Copy(C, T1)
3.    for each C_i ∈ L(C) do
4.      begin
5.        Copy(C_i, T2)
6.        R(T2) = Synthesis(R(C_i), R(I))
7.        L(T1) = L(C) − {C_i} + {T2}
8.        Score = CF(T1)
9.        if Score ≥ Max_score then
10.         begin
11.           Max_score = Score
12.           Max_concept = C_i
13.         end
14.     end
15.   return(Max_score,Max_concept)
16. end
```

TABLE I
B. CREATION_TEST

```
Procedure Creation_test(C, I)
1.  begin
2.    Copy(C, T)
3.    L(T) = L(C) + {I}
4.    return(CF(T))
5.  end
```

TABLE I
C. MERGING_TEST

```
Procedure Merging_test(C, I)
1.  begin
2.    Max_score = 0, Copy(C, T)
3.    List = Sorting(C, I)
4.    F = First_concept(List)
5.    Copy(F, M), L(M) = {F}
6.    while(TRUE)
7.      begin
8.        N = First_concept(List)
9.        L(M) = L(M) + {N}
10.       R(M) = Synthesis(R(M), R(N))
11.       L(T) = L(C) − L(M) + {M}
12.       Score = CF(T)
13.       if (Max_score ≥ Score) or (#List = 2)
14.         then
15.           return(Max_score,Max_concept)
16.         else
17.           begin
18.             Max_score = Score
19.           Copy(M, Max_concept)
20.           end
21.     end
21. end
```

n classes are extended into $n + m - 1$ if the deleted class has the number of m subclasses. Next, in the searching test, an input instance is placed in each class C_i in newly constructed class lists, and each clustering is evaluated by criterion function CF. The maximum value among these evaluated values is selected as $S_{splitting}$ and the details of the procedure are shown in Table I-D.

We give also an example to clarify the meaning of the four tests. For the splitting test, we introduce the sub-

TABLE I
D. SPLITTING_TEST

```
Procedure Splitting_test(C, C_i, I)
1.  begin
2.    Copy(C, T)
3.    L(T) = L(C) + L(C_i) − {C_i}
4.    Score, Concept = Searching_test(T, I)
5.    return(Score)
6.  end
```

(a) (b)

(a)

random element r	P_r	Q_r	Attributed random pairs
α_1	0.0	1.0	attribute1 = (X_a 1.0) attribute2 = (X_e 1.0)
α_2	0.0	1.0	attribute1 = (X_b 1.0) attribute2 = (X_e 1.0)
α_3	0.0	1.0	attribute1 = (X_b 0.4) (X_c 0.6) attribute2 = (X_d 1.0)
β_{12}	0.0	1.0	attribute3 = (X_g 1.0)
β_{13}	0.0	1.0	attribute3 = (X_f 1.0)

(b)

random element r	P_r	Q_r	Attributed random pairs
α_1	0.0	1.0	attribute1 = (X_a 1.0) attribute2 = (X_d 0.2) (X_e 0.8)
α_2	0.0	1.0	attribute1 = (X_b 1.0) attribute2 = (X_e 1.0)
α_3	0.0	1.0	attribute1 = (X_b 0.2) (X_c 0.8) attribute2 = (X_d 1.0)
β_{12}	0.0	1.0	attribute3 = (X_g 1.0)
β_{13}	0.0	1.0	attribute3 = (X_f 1.0)

Fig. 6. (a) Attributed random graph $R(C_4)$. (b) Attributed random graph $R(C_5)$.

classes C_4 and C_5 of class C_1 represented by the random graphs, $R(C_4)$ and $R(C_5)$ in Fig. 6(a) and (b), with frequency 25 and 25, respectively. The input instance I is shown in Fig. 7(a), and Fig. 7(b) shows the current concept hierarchy, Fig. 7(c)–(e) shows the concept hierarchies and evaluation scores after the input instance is placed in each class C_i. Fig. 7(f) shows the concept hierarchy and evaluation score after the creation of a new class that contains the input instance. Fig. 7(g) shows the concept hierarchy and evaluation score after class C_1 and C_2 are merged and input instance is placed in the merged class C_M. Fig. 7(h) shows the concept hierarchy and the maximum score among the criterion scores after class C_1 is split and the input instance is placed in each subclass. From Fig. 7, we found out that $S_{searching}$, $S_{creation}$, $S_{merging}$, and $S_{splitting}$ are 0.693, 0.522, 0.508, and 0.532, respectively. In this example, because $S_{searching}$ is a maximum value, the algorithm updates the concept hierarchy by sending the searching operator to the direction of class C_1.

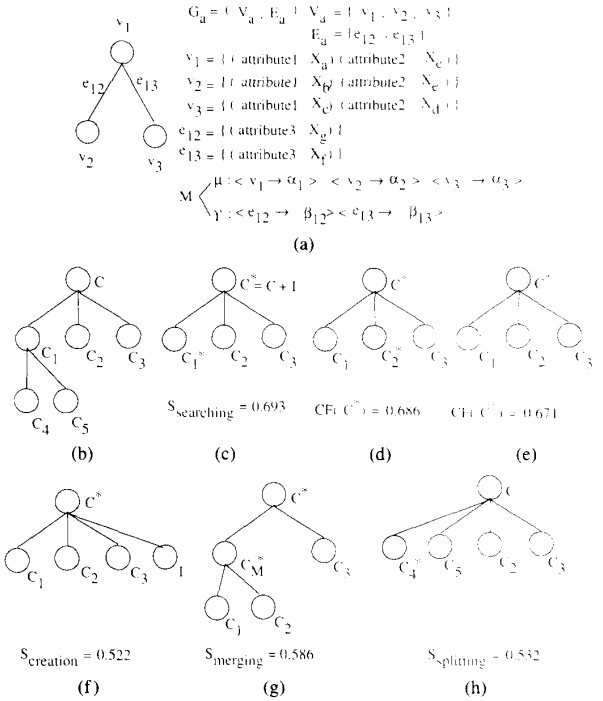


Fig. 7. (a) Input instance graph. (b) The current concept hierarchy. (c) The searching test to Class 1. (d) The searching test to Class 2. (e) The searching test to Class 3. (f) The creation test. (g) The merging test. (h) The splitting test.

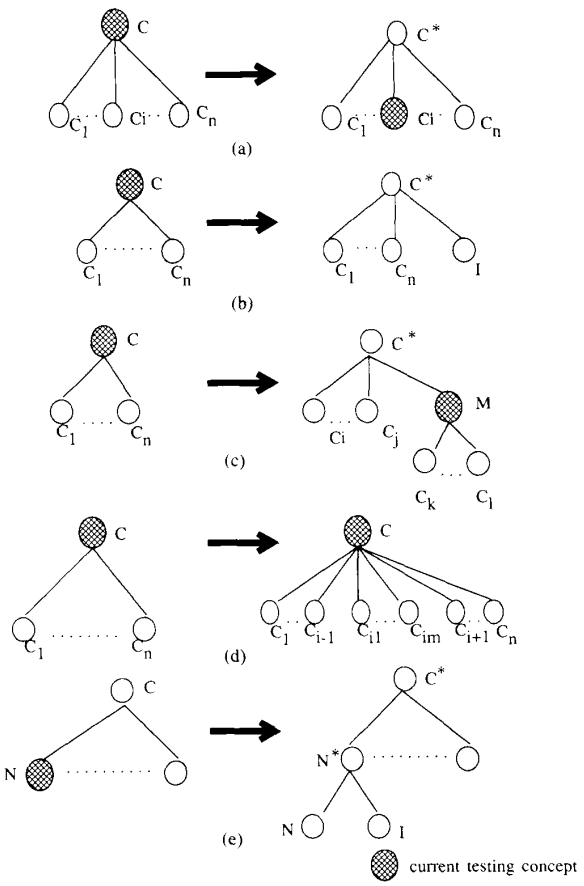


Fig. 8. Five operators. (a) Searching operator. (b) Creation operator. (c) Merging operator. (d) Splitting operator. (e) Termination operator.

C. Operators

There are five operators used in the clustering algorithm to generate a concept hierarchy as shown in Fig. 8. To sort out a new instance through a concept hierarchy and to create new disjunctive subclasses at an appropriate position in the concept hierarchy, there are searching, termination, and creation operators. Although the creation, termination, and searching operators are useful in many cases, the results of these operations are sensitive to the ordering of initial inputs. To guide against the effects of initial skewed instances, we also include the merging and splitting operators [10], [11]. A proper operator is utilized by the results of the testings to reconstruct the concept hierarchy incrementally.

- *Searching Operator:* This operator updates the selected class through synthesizing the attributed random graph of the selected class and the attributed graph of the input instance as shown in Fig. 8(a). It continues to classify the input instance down through the selected class with four tests and five operators as shown in Table II-A. In this procedure, if $S_{searching}$ is the maximum value among other testing values and the selected class is an internal concept node in the concept hierarchy, the searching operator is selected. Or, if the selected class is the terminal concept node, the termination operator is used. Otherwise, according to the $S_{creation}$, $S_{merging}$, $S_{splitting}$ value, one of the creation operators, the merging operator, and the splitting operator are selected. We use four different weighting factors such as $W_{searching}$, $W_{creation}$, $W_{merging}$, and $W_{splitting}$ for searching, creation, merging, and splitting testing values, respectively. As shown in the following section, the different distributions of this weighting factors yield somewhat different concept hierarchies. Therefore, these weighting factors can be used for adaptation to various application domains.
- *Creation Operator:* This operator creates new class and makes it a subclass of the current testing class C. In this case, the searching halts at this step, and a new class is created with the input instance as shown in Table II-B and Fig. 8(b).
- *Merging Operator:* This operator changes the structure of the concept hierarchy by merging two or more classes as shown in Table II-C and Fig. 8(c). The merging is accomplished by the synthesis of attributed random graphs corresponding to the selected classes.
- *Splitting Operator:* This operator changes the structure of the concept hierarchy by replacing the class C_i selected in the splitting test with the subclasses of this class as shown in Table II-D and Fig. 8(d).

TABLE II
A. SEARCHING_OPERATOR

```
Procedure Searching_operator(C, I, Condition)
1.  begin
2.    if Condition = TRUE
3.      R(C) = Synthesis(R(C), R(I))
4.    S_searching, S = Searching_test(C, I)
5.    S_creation = Creation_test(C, I)
6.    S_merging, M = Merging_test(C, I)
7.    S_splitting = Splitting_test(C, S, I)
8.    S_searching = W_searching × S_searching
9.    S_creation = W_creation × S_creation
10.   S_merging = W_merging × S_merging
11.   S_splitting = W_splitting × S_splitting
12.   max_value = max(S_searching, S_creation, S_merging, S_splitting)
13.     case max_value
14.       S_searching:
15.         if #S = 0
16.           then Termination_operator(C, S, I)
17.           else Searching_operator(S, I, TRUE)
18.         S_creation: Creation_operator(C, I)
19.         S_merging: Searching_operator(Merging_operator(C, M), I, TRUE)
20.         S_splitting: Searching_operator(Splitting_operator(C, S), I, FALSE)
21. end
```

TABLE II
B. CREATION_OPERATOR

```
Procedure Creation_operator(C, I)
1. begin
2.   L(C) = L(C) + {I}
3.   stop
4. end
```

TABLE II
C. MERGING-OPERATOR

```
Procedure Merging_operator(C, M)
1. begin
2.   L(C) = L(C) − L(M) + {M}
3.   return(M)
4. end
```

TABLE II
D. SPLITTING-OPERATOR

```
Procedure Splitting_operator(C, C_i)
1. begin
2.   L(C) = L(C) + L(C_i) − {C_i}
3.   return(C)
4. end
```

- *Termination Operator:* This operator creates new internal concept N by the synthesis of the concept C_i and the input instance I, and also creates a new singleton concept node with input instance I, then makes C_i, I subclasses of N. Next, the operator decides to discriminate or not the input instance from the selected concept C_i. If the criterion value of concept N is smaller than the predefined threshold, the selected concept is only updated by synthesis with the input instance. Otherwise the concept C_i in the child list $L(C)$ of concept C is replaced with a new concept N. In this case, the searching halts also at this step as shown in Table II-E and Fig. 8(e).

TABLE II
E. TERMINATION-OPERATOR

```
Procedure Termination_operator(C, C_i, I)
1. begin
2.   Copy(C_i, N)
3.   R(N) = Synthesis(R(C_i), R(I))
4.   L(N) = {C_i, I}
5.   if CF(N) ≤ Threshold
6.     then R(C_i) = Synthesis(R(C_i), R(I))
7.     else L(C) = L(C) − {C_i} + {N}
8.   stop
9. end
```

D. Algorithm

Given a new attributed graph as an instance, the clustering algorithm incorporates the instance into an existing concept hierarchy that was built from the previous instances. The result is a concept hierarchy that covers the new instance as well as previously given instances. This algorithm is started by calling the searching operator in which one among the searching operator, the termination operator, the creation operator, the merging operator, and the splitting operator is selected according to the maximum value of the $S_{searching}$, $S_{creation}$, $S_{merging}$, $S_{splitting}$ values. The classification algorithm classifies the instance through the concept hierarchy and this algorithm is similar to the clustering algorithm except that searching operator is only adopted for the classification. Because the concept hierarchy is incrementally constructed, the clustering and classification can be interleaved. That is, the concept hierarchy for the classification need not wait for all input instances, but is available after processing each instance. The clustering and classification algorithm are presented in Table III.

IV. APPLICATION EXAMPLE

To show the applicability of the proposed algorithm, we evaluate the algorithm with respect to three elements around the learning system such as the knowledge base,

TABLE III
CLUSTERING ALGORITHM AND CLASSIFICATION ALGORITHM

Incremental clustering algorithm
Input: The concept hierarchy and the input instance I.
Output: The concept hierarchy to contain the instance I.
Algorithm: *Clustering*(C_{root}, I)
1. **begin**
2. *Searching_operator*(C_{root}, I, *TRUE*)
3. **end**

Classification algorithm
Input: The concept hierarchy and the input instance I.
Output: The concept to be similar with the instance I.
Procedure *Classification*(C, I)
1. **begin**
2. *Searching_hierarchy*(C_{root}, I)
3. **end**

Procedure *Searching_hierarchy*(C, I)
1. **begin**
2. **if** $\#C = 0$
3. **then** *Show*(C), **stop**
4. **else**
5. **begin**
6. *Copy*(C, C^*)
7. $R(C^*) = Synthesis(R(C), R(I))$
8. $S_{searching}$, $S = Searching\_test(C^*, I)$
9. *Searching_hierarchy*(S, I)
10. **end**
11. **end**

the performance element, and the environment, using the simple shapes. First, the evaluation of the knowledge base shows a constructed concept hierarchy and the effects of the parameter value defined in the clustering algorithm. Second, the evaluation of the performance element covers the ability of classification and the time needed to retrieve a correct concept for a given shape. Finally, the evaluation of the environment analyzes the time needed to insert a new shape into an existing concept hierarchy.

For the structural description of an object shape, a decomposition of the shape into simpler components is the one of possible methods. Kin [8] showed that shape decomposition based on a perceptual structure can be derived from nonaccidental properties of boundary such as collinearity, regularity, symmetry, proximity, and simplicity. The components of the decomposed shape can be represented with an attributed graph. Eight different classes of toys and six different classes of planes after decomposition are listed in Fig. 9(a) and (b) and 20 instances per class are supplied.

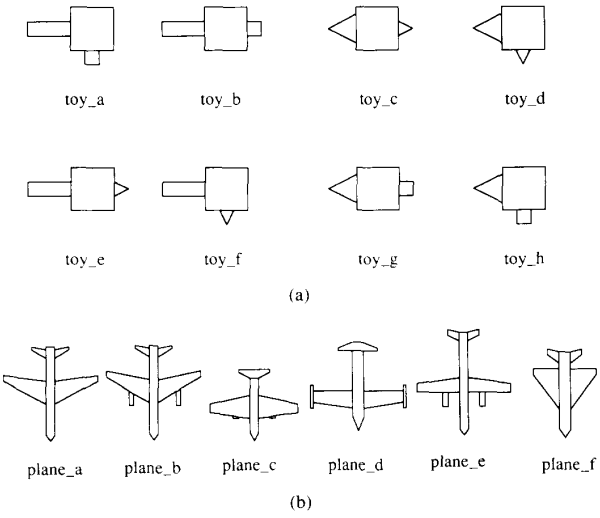


Fig. 9. (a) Toy classes. (b) Plane classes.

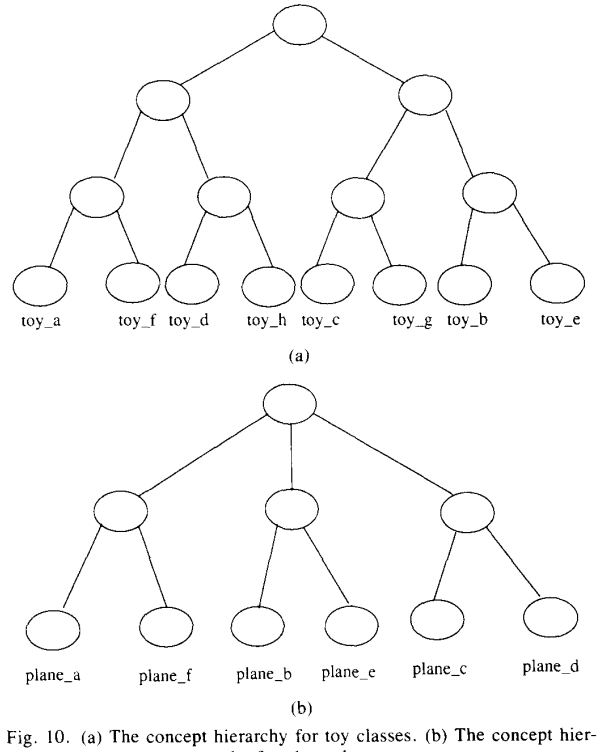


Fig. 10. (a) The concept hierarchy for toy classes. (b) The concept hierarchy for plane classes.

A. Knowledge Base

The constructed concept hierarchy for the structural input shapes is shown in Fig. 10(a) and (b). Although the class membership of given instances was not given, the resultant concept hierarchy showed that all the instances are clustered into their own class, and the classes are well grouped structurally.

For different threshold values of termination operator on the constructed concept hierarchy, we know that the number of terminal nodes in the concept hierarchy is decremented with the increment of the threshold value. In order words, the higher the parameter is, the simpler concept hierarchy constructs with more general concepts as terminal nodes. This threshold value may be used to prevent the clustering algorithm from building too large a knowledge base for real applications.

There are other parameters in our clustering algorithm for adaptation to the various application domains. $W_{merging}$, $W_{searching}$, $W_{creation}$, and $W_{splitting}$ are the weighting factors of merging, searching, creation, and splitting testing, respectively. As an experimental result, we know that

the increment of $W_{merging}$ and $W_{searching}$ provokes the increment of the average depth of terminal node in concept hierarchy. The increment of $W_{splitting}$ and $W_{creation}$ produces the increment of the average branching factor of the internal node in concept hierarchy. The proper adjustment of these parameters with the application domain should lead to both better performance and greater efficiency. From above two experiments, we know that the different distributions of parameters yield somewhat different concept hierarchies.

B. Performance Element

As a basis for the evaluation of a performance element, we take a zero value for the threshold discussed in the previous subsection and a value of $W_{searching}$, $W_{creation}$, $W_{merging}$, and $W_{splitting}$. To evaluate the quality of the constructed concept hierarchy through the classification of unknown instances, we use the remaining instances not used in clustering phase. As a result a classification rate of 100 percent was achieved thanks to structural differences between the classes, and a good clustering algorithm and criterion function. Next, we used the shapes but with components deformed by noise to retrieve the most similar concept stored in the concept hierarchy. The distortion was created by expanding or shrinking the biggest component by 10 percent. In the case of the plane, our algorithm classified 98 percent of this input's shapes, while misclassifying only 2 percent of them. Finally, we examined all the learned shapes which have a missing component caused by partial occluding. The distortion was created by deleting the smallest and terminal component. In this case, with one missing component in each plane, it correctly classified 95 percent of this input's shapes, while misclassifying only 5 percent of them.

To compute the cost of recognizing an input shape from an existing concept hierarchy, assume that B is an average branching factor of the tree, n is the number of previously classified objects, K_1 is an average cost for matching, and K_2 is an average cost for evaluation. Thus, the total number of comparisons is roughly $O(K_2 B \log_B n)$. In any case, the cost of recognizing an object in this system is significantly less expensive than systems whose database is simply a sequence of objects, where the average cost is rough $O(K_1 n)$.

C. Environment

The time needed to insert a new instance into the concept hierarchy can be calculated with respect to the number of instances presented to the system prior to the input instance, such as $K_2(4B - 2) \log_B n$, that is $O(K_2 B \log_B n)$. This insertion time includes the searching B, creation 1, merging $(B - 2)$, splitting $(2B - 1)$ at each level in the concept hierarchy in the worst case. The number of instances required to converge a stable concept hierarchy is slightly varied with the order of the given instances. However, a similar concept hierarchy is constructed regardless of the order if the instances in the class are more structurally similar to each other, rather than similar to instances in other classes.

V. Conclusion

In this paper, we have defined the criterion function based on entropy minimization for the clustering and introduced the incremental clustering system using this criterion function to group the patterns represented by attributed graphs. The task of our clustering algorithm is to take a succession of attributed graphs and to build up a concept hierarchy that summarizes and organizes the input instances. We show that the incremental clustering algorithm and classification algorithm can be interleaved from the fact that the concept hierarchy is constructed incrementally. With the attributed graph as an input instance, the clustering algorithm incrementally acquires a concept hierarchy in a top-down manner with the hill climbing strategy. The algorithm applies the best operator through states resulting from all tests using the criterion function based on entropy minimization, and iterates until there is no more progress. We demonstrated the system's capability in a structural shape recognition problem. Further research can be directed to applying the proposed approach to various applications such as automatic knowledge acquisition of expert systems and syntactic pattern recognition.

References

[1] W. H. Tsai and K. S. Fu, "Error-correcting isomorphisms of attributed relational graphs for pattern analysis," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, no. 12, pp. 757–768, Dec. 1979.

[2] ——, "Subgraph error correcting isomorphisms for syntactic pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 1, pp. 48–62, Jan. 1983.

[3] A. Sanfeliu and K. S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 3, pp. 353–362, May 1983.

[4] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-10, no. 5, pp. 695–703, Sept. 1988.

[5] A. K. C. Wong and H. E. Ghahraman, "Random graph: Structural-contextual dichotomy," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-2, no. 4, pp. 341–348, July 1980.

[6] A. K. C. Wong and M. You, "Entropy and distance of random graphs with application to structural pattern recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7, no. 5, pp. 599–609, Sept. 1985.

[7] A. K. C. Wong and S. W. Lu and M. Rioux, "Recognition and shape synthesis of 3-D objects based on attributed hypergraphs," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-11, no. 3, pp. 279–289, Sept. 1989.

[8] H. S. Kim and K. H. Park, "Shape decomposition based on perceptual structure," *Computer Vision and Image Processing*, L. Shapiro and A. Rosenfeld, Eds. New York: Academic, 1992, pp. 363–383.

[9] R. S. Wallance, "Finding natural clusters through entropy minimization," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1989.

[10] D. H. Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, vol. 2, no. 2, pp. 139–172, 1987.

[11] J. H. Gennari, P. Langley, and D. H. Fisher, "Models of incremental concept formation," *Artificial Intell.*, vol. 40, pp. 11–61, 1989.

[12] M. Lebowitz, "Experiments with incremental concept formation: UNIMEM," *Machine Learning*, vol. 2, no. 2, pp. 103–138, 1987.

[13] M. Hadzikadic and Y. Y. Yun, "Concept formation by incremental conceptual clustering," in *Proc. IJCAI*, 1988, pp. 831–836.

[14] S. Watanabe, "Pattern recognition as a quest for minimum entropy," *Pattern Recognition*, vol. 13, no. 5, pp. 381–387, 1981.

[15] R. S. Wallance and T. Kanade, "Finding natural clusters having minimum description length," in *Proc. Pattern Recognition*, pp. 438-442, 1990.
[16] S. K. Chang, *Principles of Pictorial Information System Design*. Englewood Cliffs, NJ: Prentice-Hall, 1989, ch. 4, pp. 63-65.
[17] J. G. Carbonell, "Introduction: Paradigms for machine learning," *Artificial Intell.*, vol. 40, pp. 1-9, 1989.

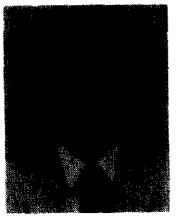
Dong Su Seong (M'89) was born in MunKyung, KyungBuk, Korea, on August 31, 1963. He received the B.S. degree in electronic engineering from the Hanyang University in 1987, and the M.S. and Ph.D. degrees in electric engineering in 1989 and August 1992, respectively, both from the Korea Advanced Institute of Science and Technology (KAIST), with a thesis on the definition of a hierarchical attributed random graph and its application to computer vision and machine learning. Since September 1992, he has been a Postdoctor with the Information and Electronic Research Institute, KAIST. His current research interests include computer vision, machine learning, and parallel processing.

Dr. Seong is a member of the Korea Information Science Society, KITE, and the IEICE.

Ho Sung Kim (M'85) was born in Seoul, Korea, on April 5, 1959. He received the B.S. degree in electronic engineering from the Hanyang University in 1982, and the M.S. and Ph.D. degrees in electric engineering in 1984 and 1988, respectively, both from the Korea Advanced Institute of Science and Technology (KAIST), with a thesis on shape representation for machine learning.

Since 1987, he has been a faculty member with the Department of Computer Science, Sungshin Womens University, where he is now Associate Professor. His current research interests include computer vision, machine learning, and cognitive science.

Dr. Kim is a member of the Korea Information Science Society, the Korea Society for Cognitive Science, and AAAI.



Kyu Ho Park (M'74) was born in Sang Ju, Korea, on October 19, 1950. He received the B.S. degree from Seoul National University, in 1973, the M.S. degree from the Korea Advanced Institute of Science and Technology (KAIST), in 1975, and the Dr. Ing. degree from the Universite de Paris, Paris, France, in 1983, all in electrical engineering.

He worked for the development of EPABX at a private company from 1975 to 1978. He was awarded a French Government Scholarship during 1979-1983 and studied at the Laboratoire des Signaux et Systems, CNRS. He is now a professor with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology. His major research interests include machine learning, computer vision, computer architecture, and parallel processing.

Dr. Park is a member of the Korea Information Science Society and KITE.