

Optimal Attribute Quantization for Mining Quantitative Association Rules

Hong Shen

School of Computing and Information Technology
Griffith University, Australia
and

Graduate School of Information Science
Japan Advanced Inst. of Sci. & Tech. Japan

Quantitative Association rule

- Notations and definitions

- A : set of items.
- $T \subset U$: transaction.
- \mathcal{D} : collection of transactions (database).
- $X \rightarrow Y$ for $X, Y \subset A$: association rule.
- δ : user-defined minimum *support*.
- μ : user-defined minimum *confidence*.

The *support* of an itemset X , $supp(X)$, is the fraction of \mathcal{D} satisfying X .

The *confidence* of rule $X \rightarrow Y$ is the ratio of $supp(X)$ to $supp(Y)$, i.e., the fraction of \mathcal{D} satisfying X that also satisfy Y .

A *strong* association rule is rule $X \rightarrow Y$, where $X, Y \subset U, X \cap Y = \emptyset$, such that $supp(X \cup Y)$ and $conf(X \rightarrow Y)$ are not smaller than predefined *minimal support* and *minimal confidence* respectively.

A can be viewed as the set of all attributes. When each attribute has only values 0 (miss) and 1 (occur) in D , the association rules generated on D are called *binary* association rules.

- The problem:

How to mine *quantitative* association rules from databases having numerical and/or categorical attributes rather than boolean attributes. These rules contain quantitative values of the numerical/categorical attributes.

E.g. $\in [40, 50] \wedge \text{Married} \rightarrow \text{Cars} = 2$.

Existing Solutions

- Attribute value enumeration

Enumerate different categorical states and numerical values and treat them as items. Then use a boolean association rule mining algorithm to find all rules. Finally group rules with adjacent numerical values together.

- Attribute discretization

Map quantitative attributes onto boolean attributes and then apply boolean association rule mining. This consists of

- partitioning the continuous attributes to sets of intervals (*attribute discretization*),
- mapping intervals to items, and
- applying boolean association rule mining.

Attribute Discretization Methods

1. Supervised discretization:
Need pre-specified optimal classes. Not suitable for association mining.
2. Unsupervised discretization:
 - Equal-width discretization: Divide the continuous attribute range into N intervals of equal width.
 - Equal-depth (or equal-cardinality) discretization: Divide the continuous attribute range into N intervals so that there are $1/N$ of the total instances in each interval.

- Problems with unsupervised discretization

- Equal-width may lead to too few values in some intervals and too many in others, both cases hinder mining high quality association rules.
- Equal-depth may separate similar attribute values into different intervals and group dissimilar attribute values into the same interval.

The main cause of these problems is that either method does not consider both value density and distance at the same time.

- Adaptive merge criterion

Consider both interval width (value distance) and depth (value density) during the course of attribute discretization to produce intervals with proper value density and distance so that association rules can be easily found from them.

The criterion for merging adjacent intervals that combines interval width and depth takes into consideration of the following observations:

Adaptive merge works as follows:

Initially place each numerical attribute value in a separate interval. Then use the above merge criterion to selectively merge similar adjacent intervals based on average intra-interval distance.

Merge on average distance

The pair of adjacent intervals with the smallest average difference among a collection of intervals is chosen to be merged. This process is repeated until either the density of each interval is large enough to form a rule, or each pair of adjacent intervals are so far apart that they are unlikely to be placed in one group.

This algorithmmerge requires $O(m^2)$ time.

Optimal Merge Algorithm

- Maximal intra-interval distance

Notations and Assumptions:

- Numerical attribute I has m distinct values, $I = \{x_0, x_1, \dots, x_{m-1}\}$, where attribute value x_i has n_i occurrences in the database (weight), $x_i < x_{i+1}$ for all $0 \leq i \leq m-2$, thus $N = \sum_{i=0}^{m-1} n_i$ attribute value occurrences.
- $I_{j_0}, I_{j_1}, \dots, I_{j_k}$ is a set of maximal disjoint intervals on I , where initially $j_i = i$, $k = m-1$, and I_{j_i} contains x_i , and *representative center* c_i equals x_i .

Given $I_u = \{x_u, \dots, x_{v-1}\}$ and $I_v = \{x_v, \dots, x_{w-1}\}$, where I_t has rep center c_t and N_k attribute value occurrences (size), their merged interval $I'_u = I_u \cup I_v$ has new rep center

$$c'_u = \frac{c_u N_u + c_v N_v}{N_u + N_v},$$

and size

$$N'_u = N_u + N_v = \sum_{i=u}^{w-1} n_i.$$

The *maximal intra-interval distance* of interval I_t wrt its representative center c_t is defined by:

$$D^*(I_t, c_t) = \max_i |x_i - c_t|. \quad (1)$$

The argument for using maximal distance is that instances at a greater distance to the interval's representative center are less similar to the center than those at a smaller distance, and will not be covered by rules derived in the interval if this distance exceeds a certain level.

An interval merge scheme is *feasible* if it results in a minimum number of intervals whose maximal intra-interval distances are each within a given threshold and populations are as equal as possible.

- Algorithm: Optimal merge on maximal distance

For k intervals, let the average population of each interval be $\bar{N}_k = N/k$ and the population *deviation* of interval I_u be $\Delta_u = |N_u - \bar{N}_k|$, where N_u is the actual population of I_u . Initially, $I_u = \{x_u\}$ and $0 \leq u \leq m - 1$. The algorithm consists of two steps below:

Merge

1. Partition $\mathcal{I} = \{I_0, I_1, \dots, I_{m-1}\}$ into a minimum number of intervals such that each interval has a maximal intra-interval distance not greater than d ;
2. Assume the above step produces k intervals: $\{I_{u_0}, I_{u_1}, \dots, I_{u_{k-1}}\}$, $0 = u_0 < u_1 < \dots < u_{k-1} \leq m - 1$. For $I_{u_j} = [X_{u_j} : c_{u_j}]$, where $X_{u_j} = \{x_{u_j}, x_{u_j+1}, \dots, x_{u_{j+1}-1}\}$ and c_{u_j} is the representative center of I_{u_j} , check if moving $x_{u_{j+1}-1}$ to $I_{u_{j+1}}$ will result in better load balance while still preserving the maximal intra-interval distance property, and do it if will.

Step 1 can be implemented by *linear-scan* to form appropriate segments of intervals after a single pass: starting from I_0 merge I_u with I_{u+j} for $j = 1, 2, \dots$, until the maximal intra-interval distance not smaller than the threshold; continue this until there is no interval remaining to be merged. This process requires time $O(m)$.

Step 2 examines every adjacent pair of intervals after merge, requiring at most $m - 1$ steps. Each step checks the changes of population deviation by moving $u_{j+1} - 1$ instances from I_{u_j} to $I_{u_{j+1}}$. Do the move only when the deviation reduces. This step requires $O(m)$ time as well.

- Algorithm Merge is feasible: Linear-scan merge produces a minimum number of intervals on \mathcal{I} such that each interval has an maximal intra-interval distance not greater than d . Step 2 ensures load balancing on population.
- Algorithm Merge is time optimal: $\Omega(m)$ is a lower bound for interval merging.

Parallel implementation

Assume that we are given a parallel machine with p processors and originally all intervals are evenly distributed over these processors.

ParMerge

1. P_i calls algorithm Merge to merge the m/p intervals allocated to it and take the first and last intervals (I_i, I'_i) for further merge with other processors in the next step, for all $0 \leq i \leq p - 1$ in parallel;
2. Repeat the following $\log p$ times for all active P_i , $0 \leq i \leq p - 2$, in parallel:
{*Assume the indices of intervals are still contiguous after merge, for simplicity in description.*}
 - (a) If the maximal intra-interval distance after merge of I'_i and I_{i+1} is not greater than threshold d , P_i writes “ Δ_i ” in f_i , where Δ_i is the population difference after merge; otherwise writes ∞ to f_i ;
 - (b) If $f_i = \infty$, P_i marks itself *inactive*;

- (c) If $i \leq p - 2$ is even and $f_i \leq f_{i+1}$, P_i merges I'_i and I_{i+1} , and updates f_i to be either *infity* or the population deviation of I'_i and the next interval after merge as stated in Step 1;
 {*Even index interval merging, where two consecutive pairs both can be merged merge the pair with smaller population deviation for load balance.*}
- (d) If $i \leq p - 2$ is odd and $f_i \leq f_{i+1}$, P_i merges I'_i and I_{i+1} , and updates f_i to be either ∞ or the population deviation of I'_i and the next interval after merge;
 {*Odd index interval merging, where two consecutive pairs both can be merged merge the pair with smaller population deviation for load balance.*}

Algorithm ParMerge runs in $O(\frac{m}{p} + T_d \log p)$ time using p processor on EREW PRAM.

Quantitative Association Rule Mining

Our quantitative association rule mining algorithm consists of the following three steps.

1. Numerical attribute discretization

Find a set of suitable intervals using the above algorithm and then map all numerical values in each interval onto one item.

2. Frequent itemsets finding

Use the following set-trie based boolean association rule mining algorithm to find all frequent itemsets:

- (a) Initiate set trie: Find all *frequent 1-itemsets* and *frequent 2-itemsets*, and place them on layer 1 and 2 respectively in a *set trie*. Label each node in the set trie with the last (largest) item of its itemset.
- (b) Generate candidates for frequent *itemsets* in the set trie: For node i_{k_j} containing $\{i_1, i_2, \dots, i_{k_j}\}$ with k_m sibling nodes at layer k , $k = 3, 4, \dots$, it take node $\{i_1, i_2, \dots, i_{k_j}, i_{k_q}\}$ as a child node

at layer $k + 1$ for every $k_j < k_q \leq k_m$ if (i_{k_j}, i_{k_q}) is an frequent 2-itemset.

- (c) Count support of candidates: For each transaction T in the database, trace the set trie level by level and add 1 to the counter of each node $i \in T$.
 - (d) Delete infrequent itemsets: For each node in the set trie delete the whole subtree rooted at it if its counter value versus the database size is smaller than the minimum support.
3. Rule forming: Test whether two disjoint subsets of an frequent itemset can form a (strong) rule by checking the confidence of the rule, and keep the rule if it is strong.
 4. Rule pruning: Delete redundant rules from the rule set. If several rules cover the same data set, select the one with highest confidence and remove the rest. When a rule $X \rightarrow Y$ is selected, all transactions $T \supset X \cup Y$ in \mathcal{D} will be removed and the confidences of the rest rules will be updated accordingly. Repeat this process until all transactions in \mathcal{D} are removed.

An example set trie:

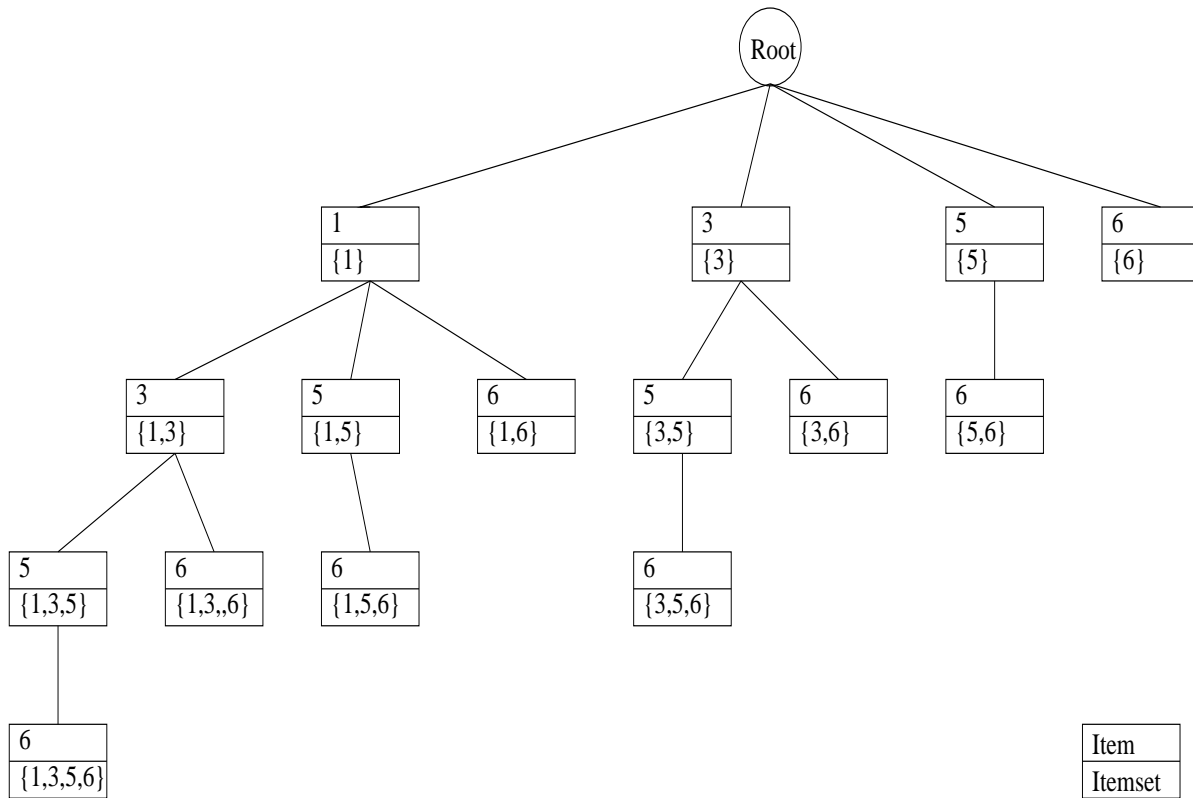


Figure 1: An example of set trie

Concluding Remarks

- We presented an optimal numeric attribute quantization algorithm based on maximal intra-interval distances that generalize both equal-width and equal-depth methods.

We have shown our algorithm can be implemented in parallel cost-optimally.

- Future work includes extending the proposed method

to incorporate other measures on quality of discretization, and apply it for discretization of continuous attributes in clustering problems.

Evaluation Criterion

The covering data set of a rule set $R = \{X_i \rightarrow Y, i = 1, 2, \dots, m\}$ over database $\mathcal{D} = \{T_1, T_2, \dots, T_n\}$ is defined as:

$$C(R) = \bigcup_{i=1}^m T_{k_i},$$

where $(X_k \cup Y) \subset T_{k_i}$ for $1 \leq k \leq m$.

For several rule sets $R_j = \{X_i \rightarrow Y_j, i = 1, 2, \dots, m_j\}$, $j = 1, 2, \dots, k$, where $\bigcap_{j=1}^k Y_j = \emptyset$, the coverage of rule set $\bigcup_{j=1}^k R_j$ over database \mathcal{D} is defined as:

$$Cov\left(\bigcup_{j=1}^k R_j\right) = \frac{1}{|\mathcal{D}|} \sum_{j=1}^k |C(R_j)|$$

If numerical attributes are partitioned suitably, then the resulting rule set has a large coverage. Otherwise, the coverage is low.

- Brief description of databases used in experiment

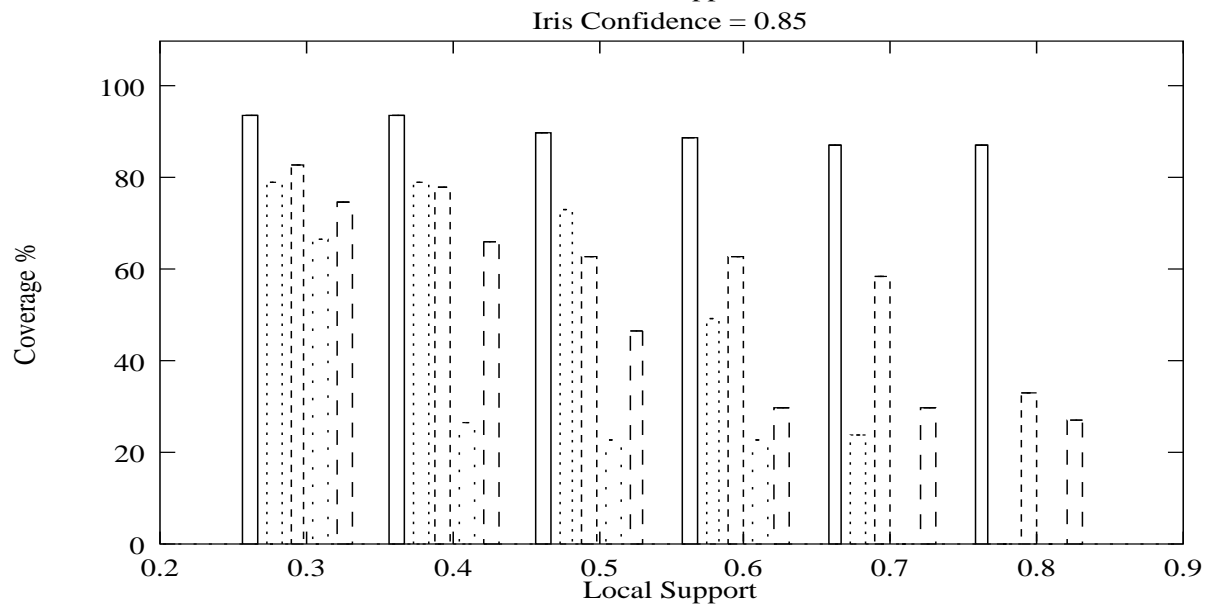
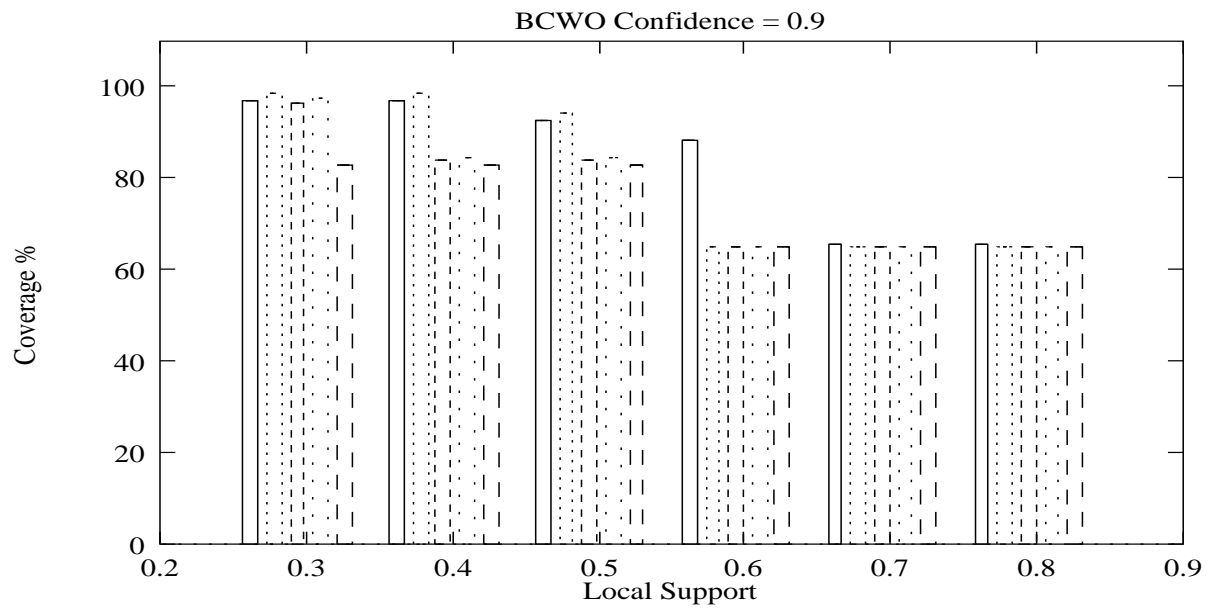
Database name	Record	Attribute	Class
Glass Identification	214	9 Numerical	3
Heart Disease	270	7 Num. + 6 Cat.	2
Iris Plant	150	4 Numerical	3
Wisconsin Breast Cancer	699	10 Numerical	2

Table 1: Brief description of databases

Experimental Results

We compared Algorithm 1 with equal-width and equal-depth discretization methods, where numerical attributes were partitioned into 5 or 10 equal value intervals or equal density intervals respectively, and they are denoted as equal-width 5, equal-width 10, equal-depth 5 and equal-depth 10.

In total 24 trials, only 1 trial is worse, 4 trials are marginally lower, and the other 19 trials are better than or equal to equal-width and equal-depth methods.



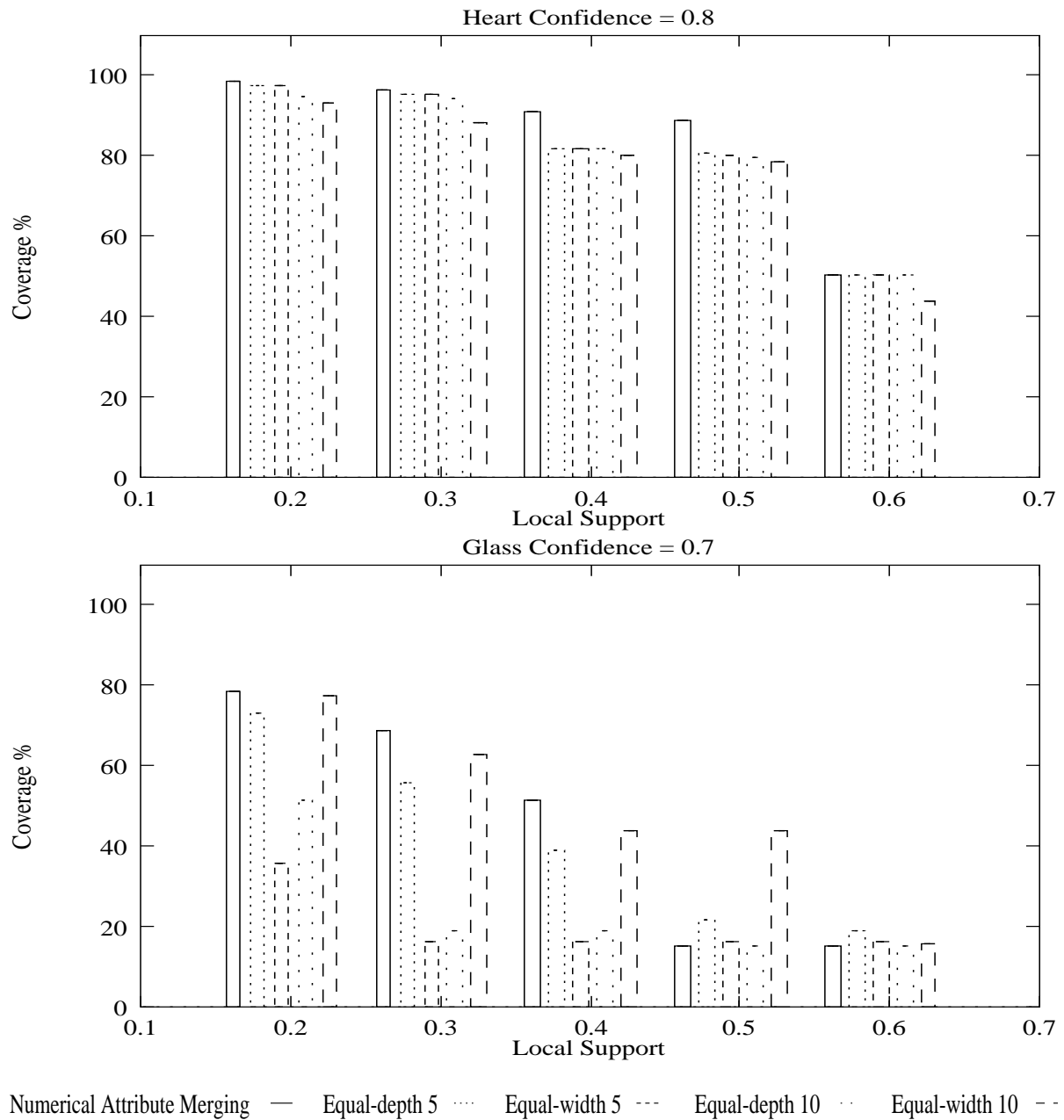


Figure 2: Comparison of mining result of three class blind methods