Semantic Compression
Semantic Compression

- Real-life data are highly structured and there are strong correlations between the attributes and records.
- The *syntactic* compression algorithms are not designed to take advantage of such structures.
- Recently, some new schemes are proposed based on these observations.
- Naturally, such schemes are so-called *semantic* compression methods in contrary to the *syntactic* ones.
Semantic Compression

- First derive a description model by taking into account the semantic meaning of the attributes.
- Represent the original data by the derived model.
- The data, that cannot derive from the model, are explicitly stored.
Syntactic Compression

- Statistical Model vs. Dictionary-based Model
  - Statistical Model
    - Each distinct character of the input data is encoded with the code assignment being based on the probability of the character’s appearance in the data.
    - E.g. Huffman coding, Arithmetic coding
  - Dictionary-based Model
    - Maintains a dictionary that contains a list of commonly occurring character strings in data and their corresponding codes
    - E.g. LZW, Vector Quantization

- Lossless vs. Lossy
  - Lossless: Huffman coding, Arithmetic coding, LZW coding
  - Lossy: Vector quantization
Syntactic Compression

Huffman Coding
- First developed by David Huffman.
- Symbols that have higher probabilities will have shorter codes than symbols that have lower probabilities.
- The two symbols that have minimum probabilities will have the same length.

LZW (Lempel-Ziv-Welch) coding
- Used both in UNIX compress and DOS pkzip
- Organized around a string translation table which contains a set of character strings and their corresponding code values
- The string table has prefix property that for every string in the table, its prefix is also in the table.
Lossless Compression (static)

- **Dictionary Encoding**
  - Assigns an ID to each new word
  - Input: ABC  ABC BC DDD
  - Compressed Data:  1 1 2 3
  - Dictionary: ABC =1, BC = 2, DDD=3

- **Binary Encoding**
  - Binary representation of numeric data
  - Input: “100” “20” “50”
  - Encoding: 100 20 50
Lossless Compression (static)

- Differential Encoding (or Delta Encoding)
  - Replaces a data item with a code value that defines its relationship to a specific data item

ex)
input: 100 120 130
Compressed Data: 100 20 30

input: Johnson Jonah Jones Jorgenson
Compressed Data: (0) Johnson (2)nah (3)es (2)rgenson
Lossless Compression (semi-adaptive)

- Huffman Encoding
  - Assigns shorter codes to more frequently appearing symbols and longer codes to less frequently appearing symbols
  - ex)
  - input: ACE
  - Encoding: 01001101

Huffman tree
Lossless Compression (adaptive)

- **LZ(Lempel–Ziv) Coding**
  - Adaptive dictionary encoding
  - Converts variable-length strings into fixed-length codes

Input: \{A B AB AA ABA\}
Compressed Data: \{(0,A)(0,B)(1,B)(1,A)(3,A)\}
- new table entry is coded as \((i,c)\)
  - \(i\): the codeword for the existing table entry (12 bit)
  - \(c\): the appended character (8 bit)
Fascicle

[Jagadish, Madar, Ng 99]

Fascicles

- Informally, subsets of a relation having very similar values for many attributes
- Technically, a $k$–D fascicle of a relation is a subset of records having $k$ compact attributes

An attribute $A$ of a subset $F$ of records is compact with tolerance $t$, if:

- the range of $A$–values (numeric), or
- the number of distinct $A$–values (categorical) of all the records in $F$ does not exceed $t$
What is a Fascicle? (cont.)

- Compress data by storing representative values (e.g., “centroid”) 
  *only once* for each attribute cluster

- *Lossy* compression: information loss is controlled by the notion of 
  “similar values” for attributes (*user defined*)

<table>
<thead>
<tr>
<th>age</th>
<th>salary</th>
<th>assets</th>
<th>credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30,000</td>
<td>25,000</td>
<td>good</td>
</tr>
<tr>
<td>30</td>
<td>35,000</td>
<td>50,000</td>
<td>good</td>
</tr>
<tr>
<td>35</td>
<td>40,000</td>
<td>75,000</td>
<td>good</td>
</tr>
<tr>
<td>40</td>
<td>100,000</td>
<td>175,000</td>
<td>poor</td>
</tr>
<tr>
<td>50</td>
<td>110,000</td>
<td>250,000</td>
<td>good</td>
</tr>
<tr>
<td>60</td>
<td>50,000</td>
<td>150,000</td>
<td>poor</td>
</tr>
<tr>
<td>70</td>
<td>35,000</td>
<td>125,000</td>
<td>poor</td>
</tr>
<tr>
<td>75</td>
<td>15,000</td>
<td>100,000</td>
<td>good</td>
</tr>
</tbody>
</table>

- 3-D Fascicles
  - 35,000 50,000 good :3

- 2-D Fascicles
  - 137,500 poor :2
Fascicle

- **Lossless**:  
  - First, use fascicles to physically re-order the relation  
  - Compact attributes are *not* projected away  
  - Apply syntactic compression  
    - Syntactic compression dependent on the physical ordering of records

- **Lossy**:  
  - First, use fascicles and project away compact attributes  
  - Apply syntactic compression
Fascicle

dataset: AT&T

- syntactic only
- fascicle re-ordering+syntactic
- fascicle lossy+syntactic

lossless: extra 25% compression
lossy: extra 75% compression (max error $t = 1/32$)

From [Jagadish, Madar, Ng 99]
Compressing with Fascicles

- k-dimensional fascicle $F(k,t)$: subset of records with $k$ compact attributes
  - Compress by storing single centroid value for $k$ compact attributes
- User-defined compactness tolerance $t$ (vector) specifies the allowable loss in the compression per attribute
  - E.g., $t[\text{Duration}] = 3$ means that all “Duration” values in a fascicle are within 3 of the centroid value
Compressing with Fascicles

- Problem Statement
  - Given a table \( T \) and a compactness-tolerance vector \( t \),
  - Find fascicles within the specified tolerances such that the total storage is minimized (so-called ‘storage minimization problem’)

- Problem Decomposition
  1. Find candidate fascicles in \( T \)
  2. Select the best fascicles to compress \( T \)

- NP-Complete
  - Corresponds to Minimum Cover Problem \([Karp 72]\)
Storage Minimization Problem

- Given a collection $C$ of subsets of a finite set $S$ and a positive integer $K$,
- Is there a subset $C' \subseteq C$ with $|C'| \leq K$ such that every element of $S$ belongs to at least one member of $C'$?
- NP-Complete
- Greedy selection is among the best existing heuristics
How to Find Candidate Fascicles?

- Operates on the lattice consisting of all possible subsets of records
- Finding all fascicles needs too MUCH effort
- Greedy selection only needs some good quality candidates, not all of them
- Thus, we adapt randomized strategy
  - Pick some good starting fascicles
  - Grow them to maximal sizes to ensure quality by one scan over data
Tip set & Maximal set

- **Tip set**
  - It is hard to find the exact $k$-D fascicles.
  - To find a $k$-D fascicle for a given value $k$.
  - $(\perp \subseteq S_1 \subseteq S_2 \subseteq S_3 \subseteq \ldots \subseteq \top)$
    - $\top$: entire relation, $\perp$: empty set
    - $S_t$: $i$-D fascicle
    - $S_{t+1}$: $j$-D fascicle, a superset of $S_t$
    - Then, $j \leq i$
    - For $1 \leq k \leq n$, if $j < k \leq i$, we call $S_t$ a tip set.
  - In other words, $S$ is a $k$-D tip set if $S$ is a $k$-D fascicle, and there is an parent $T$ of $S$ such that $T$ is a $j$-D fascicle with $j < k$.

- **Maximal set**
  - $S$ is a $k$-D maximal set if $S$ is a $k$-D fascicle, and for all supersets $T$ of $S$, $T$ is a $j$-D fascicle with $j < k$
Why tipset? – Example

- We want 2-D fascicle.
- We add one more tuple to 4-D fascicle.
- But, it becomes 1-D fascicle.

4-D fascicle

<table>
<thead>
<tr>
<th>May</th>
<th>Winger</th>
<th>35</th>
<th>290</th>
<th>180</th>
</tr>
</thead>
</table>

1-D fascicle

<table>
<thead>
<tr>
<th>May</th>
<th>Winger</th>
<th>35</th>
<th>290</th>
<th>180</th>
</tr>
</thead>
<tbody>
<tr>
<td>Odjick</td>
<td>Winger</td>
<td>9</td>
<td>115</td>
<td>245</td>
</tr>
</tbody>
</table>
Algorithm Single-k

- Input: A dimensionality k, number of fascicles P, a buffer of b pages, and a relation R of r pages
- Output: P k-D fascicles

1. Divide R into q disjoint pieces, each comprising up to b randomly chosen pages from R, i.e., $q = \lceil r/b \rceil$.
2. For each piece: /* choosing initial tip sets */
   2.1 Read the piece into main memory.
   2.2 Read the records in main memory to produce a series of tip sets.
   2.3 Repeat 2.2, each with a different permutation of the records, until $P/q$ tip sets are obtained.
3. /* growing the tip sets */
   Grow all P tip sets with one pass over the relation.
   Output the grown tip sets.
Single-\(k\) algorithm – Example

- We want 2-D fascicles
  compactness tolerance \(t_{\text{Position}} = 1, t_{\text{Points}} = 10, t_{\text{Played Mins}} = 60, t_{\text{Penalty Mins}} = 20\)

<table>
<thead>
<tr>
<th>Name</th>
<th>Position</th>
<th>Points</th>
<th>Played Mins</th>
<th>Penalty Mins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blake</td>
<td>Defense</td>
<td>43</td>
<td>395</td>
<td>34</td>
</tr>
<tr>
<td>Borque</td>
<td>Defense</td>
<td>77</td>
<td>430</td>
<td>22</td>
</tr>
<tr>
<td>Cullimore</td>
<td>Defense</td>
<td>3</td>
<td>30</td>
<td>18</td>
</tr>
<tr>
<td>Gretzky</td>
<td>Defense</td>
<td>130</td>
<td>458</td>
<td>26</td>
</tr>
<tr>
<td>Konstantinov</td>
<td>Defense</td>
<td>10</td>
<td>560</td>
<td>120</td>
</tr>
<tr>
<td>May</td>
<td>Winger</td>
<td>35</td>
<td>290</td>
<td>180</td>
</tr>
<tr>
<td>Odjick</td>
<td>Winger</td>
<td>9</td>
<td>115</td>
<td>245</td>
</tr>
<tr>
<td>Tkachuk</td>
<td>Centre</td>
<td>82</td>
<td>530</td>
<td>160</td>
</tr>
<tr>
<td>Wotton</td>
<td>Defense</td>
<td>5</td>
<td>38</td>
<td>6</td>
</tr>
</tbody>
</table>

- Green cells represent that the attributes are compact.
- Red cells represent that the attributes is not compact.
- Black cells represent that the attributes need not check because those cells were red cells in previous step.
Greedy Selection for the Single-\(k\) algorithm

- To represent the storage savings induced by a fascicle \(F\), it is weighted by \(\text{wt}(F) = k \times |F|\), where \(k\) is the dimensionality of \(F\).
- In a straightforward implementation of the greedy selection, we select the candidate fascicle with the highest weight.
  - adjust the weight of the remaining fascicles.
- If \(A\) is selected fascicle, then the adjusted weight of each remaining fascicle \(F\) is given by
  \[\text{Wt}(F/A) = k \times |F - A|\]
- Then from among the remaining fascicles, we pick the one with the heaviest adjusted weight, and repeat.
The multi-\( k \) Algorithm

- Exploits single-\( k \) algorithm to produce fascicles all having dimensionalities \( \geq k \).
- Recall from the Single-\( k \) algorithm how a k-D tip set corresponds to a path \((\bot, S_1, S_2, S_3, S_t)\).
- While Single-\( k \) algorithm construct a path \((\bot, S_1, S_2, S_3, S_t)\) and obtains \( S_t \) as a k-D tip set,
  - the Multi-\( k \) algorithm uses exactly the same path to obtain larger sets on the path with dimensionality \( i \), for \( i \geq k \).
Classification
Classification

- Given:
  - Database of tuples, each assigned a class label
  - Develop a model/profile for each class
    - Example profile (good credit):
      - \((25 \leq \text{age} \leq 40 \text{ and income} > 40k) \text{ or } (\text{married} = \text{YES})\)

- Sample applications:
  - Credit card approval (good, bad)
  - Bank locations (good, fair, poor)
  - Treatment effectiveness (good, fair, poor)
What is Classification?

- Given a database of tuples
  - Each tuple consists of
    - A set of Attribute values
    - A assigned categorical class label
  
- Develop a model/classifier for each class based on the set of attributes

- Use the model to predict the class label of future data
Classification Model

- Decision Tree Model
- Probabilistic Model (Bayesian etc.)
- Neural Network Model
- Support Vector Machine
- K–nearest neighbor
Decision Trees

<table>
<thead>
<tr>
<th>salary</th>
<th>education</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>high school</td>
<td>reject</td>
</tr>
<tr>
<td>40000</td>
<td>under graduate</td>
<td>accept</td>
</tr>
<tr>
<td>15000</td>
<td>under graduate</td>
<td>reject</td>
</tr>
<tr>
<td>75000</td>
<td>graduate</td>
<td>accept</td>
</tr>
<tr>
<td>18000</td>
<td>graduate</td>
<td>accept</td>
</tr>
</tbody>
</table>

Credit Analysis

- **salary < 20000**
  - **yes**: education in graduate
    - **yes**: accept
    - **no**: reject
  - **no**: accept

- **education in graduate**
  - **yes**: accept
  - **no**: reject
Decision Trees

- **Pros**
  - Fast execution time
  - Generated rules are easy to interpret by humans
  - Scale well for large data sets
  - Can handle high dimensional data

- **Cons**
  - Cannot capture correlations among attributes
  - Consider only axis-parallel cuts
Decision Tree Algorithms

- Classifiers from machine learning community:
  - ID3[Qui86]
  - C4.5[Qui93]
  - CART[BFO84]
- Classifiers for large database:
  - SLIQ[MAR96], SPRINT[SAM96]
  - SONAR[FMMT96]
  - Rainforest[GRG98]

- Pruning phase followed by building phase
A decision tree is created in two phases:

**Building Phase**
- Recursively split nodes using best splitting attribute for node until all the examples in each node belong to one class

**Pruning Phase**
- Prune leaf nodes recursively to prevent over-fitting
- Smaller imperfect decision tree generally achieves better accuracy
SPRINT

- [Shafer, Agrawal, Manish 96]
- Building Phase
  - Initialize root node of tree
  - while a node N that can be split exists
    - for each attribute A, evaluate splits on A
    - use best split to split N
  - Use gini index to find best split
  - Separate attribute lists maintained in each node of tree
  - Attribute lists for numeric attributes sorted
How can we get best split?

- Select the attribute that is most useful for classifying training set
- Gini index and entropy
  - Statistical properties
  - Measure how well an attribute separates the training set
    - Entropy ( \( \text{entropy}(T) = - \sum p_j \times \log_2(p_j) \) )
    - Gini Index ( \( \text{gini}(T) = 1 - \sum p_j^2 \) )
Entropy example

<table>
<thead>
<tr>
<th>salary</th>
<th>education</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>high school</td>
<td>reject</td>
</tr>
<tr>
<td>40000</td>
<td>under graduate</td>
<td>accept</td>
</tr>
<tr>
<td>15000</td>
<td>under graduate</td>
<td>reject</td>
</tr>
<tr>
<td>75000</td>
<td>graduate</td>
<td>accept</td>
</tr>
<tr>
<td>18000</td>
<td>graduate</td>
<td>accept</td>
</tr>
</tbody>
</table>

\[
Entropy(S_{left}) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3}
= 0.918296
\]

\[
Pobablity(class = "reject") = \frac{2}{5}
\]

\[
Pobablity(class = "accept") = \frac{3}{5}
\]

\[
Entropy(S) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5}
= 0.970951
\]

\[
Entropy_{split}(S) = \frac{3}{5} \times 0.918296 + \frac{2}{5} \times 0 = 0.550978
\]

salary < 20000

yes

no

Entropy(S_{right}) = 0
**Pruning Phase**

- Smaller imperfect decision tree generally achieves better accuracy
- Prune leaf nodes recursively to prevent over-fitting

```
[# of tuples : 20000]

yes  no

 [# of tuples : 20001]

yes  no

[# of tuples : 20000]  [# of tuples : 1]

yes  no
```
SPRINT creates an attribute list for each attribute
Numerical attribute list is sorted
Attribute records contains
  Attribute value
  Class label
  Index of the record

<table>
<thead>
<tr>
<th>salary</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>15000</td>
<td>accept</td>
<td>2</td>
</tr>
<tr>
<td>18000</td>
<td>reject</td>
<td>4</td>
</tr>
<tr>
<td>40000</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>75000</td>
<td>accept</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>education</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>high school</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>under graduate</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>under graduate</td>
<td>reject</td>
<td>2</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>3</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>4</td>
</tr>
</tbody>
</table>

[Attribute list for salary] [Attribute list for education]
All attribute lists are made at the root

As the tree is grown, the attribute lists belonging to each node are partitioned and associated with the children

<table>
<thead>
<tr>
<th>salary</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>15000</td>
<td>accept</td>
<td>2</td>
</tr>
<tr>
<td>18000</td>
<td>reject</td>
<td>4</td>
</tr>
<tr>
<td>40000</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>75000</td>
<td>accept</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>education</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>high school</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>under graduate</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>under graduate</td>
<td>reject</td>
<td>2</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>3</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>education</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>high school</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>under graduate</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>3</td>
</tr>
</tbody>
</table>

salary < 20000

yes

no
For continuous attributes, two histograms are associated with each decision-tree node. These histograms, denoted as $C_{above}$ and $C_{below}$:

- $C_{below}$: maintains this distribution for attribute records that already been processed
- $C_{above}$: maintains this distribution for attribute records that have not been processed
Finding Split Points

- Numeric attributes
  - $C_{\text{below}}$ initials to zeros
  - $C_{\text{above}}$ initials with the class distribution at that node
  - Scan the attribute list to find the best split

- Categorical attributes
  - Scan the attribute list to build the count matrix
  - Use the subsetting algorithm to find the best split
Evaluate numeric attributes

![Histogram for salary]

- **Position 1**
  - \( C_{\text{below}} = \begin{bmatrix} 0 & 1 \end{bmatrix} \)
  - \( C_{\text{above}} = \begin{bmatrix} 3 & 1 \end{bmatrix} \)
  - \( \text{Entropy}_{\text{split}} (S) = \frac{1}{5} \times \frac{1}{1} \log 1 + \frac{4}{5} \times \left( -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} \right) \)
  - \( = 0.811278 \)

- **Position 2**
  - \( C_{\text{below}} = \begin{bmatrix} 1 & 1 \end{bmatrix} \)
  - \( C_{\text{above}} = \begin{bmatrix} 2 & 1 \end{bmatrix} \)
  - \( \text{Entropy}_{\text{split}} (S) = 0.950978 \)

- **Position 3**
  - \( C_{\text{below}} = \begin{bmatrix} 1 & 2 \end{bmatrix} \)
  - \( C_{\text{above}} = \begin{bmatrix} 2 & 0 \end{bmatrix} \)
  - \( \text{Entropy}_{\text{split}} (S) = 0.550978 \)

- **Position 4**
  - \( C_{\text{below}} = \begin{bmatrix} 2 & 2 \end{bmatrix} \)
  - \( C_{\text{above}} = \begin{bmatrix} 1 & 0 \end{bmatrix} \)
  - \( \text{Entropy}_{\text{split}} (S) = 0.8 \)

Choose Position 3 has lowest Entropy!
Evaluate categorical attributes

[Attribute List]

<table>
<thead>
<tr>
<th>education</th>
<th>label</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>high school</td>
<td>reject</td>
<td>0</td>
</tr>
<tr>
<td>under graduate</td>
<td>accept</td>
<td>1</td>
</tr>
<tr>
<td>under graduate</td>
<td>reject</td>
<td>2</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>3</td>
</tr>
<tr>
<td>graduate</td>
<td>accept</td>
<td>4</td>
</tr>
</tbody>
</table>

[Histogram for education]

<table>
<thead>
<tr>
<th></th>
<th>accept</th>
<th>reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>high school</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>under graduate</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>graduate</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

3 distinct value $\rightarrow 2^3-2$ split condition exists!

\[
\text{Entropy} \quad S(E) = \frac{1}{5} \times \frac{1}{4} \log \frac{1}{4} + \frac{4}{5} \times \left( -\frac{3}{4} \log \frac{3}{4} - \frac{1}{4} \log \frac{1}{4} \right) = 0.811278
\]

\[
\text{Entropy}_{\text{split}} (S) = 0.950978
\]

\[
\text{Entropy}_{\text{split}} (S) = 0.550978
\]

\[
\text{Entropy}_{\text{split}} (S) = 0.811278
\]

\[
\text{Choose} \left\{ \text{graduate} \right\} \text{ has lowest Entropy!}
\]
[S. Babu, M. N. Garofalakis, and R. Rastogi 01]

Model-Based Semantic Compression (MBSC)
- Extract *Data Mining models* from the data table
- Use the extracted models to construct an effective compression plan
- Lossless or lossy compression

*SPARTAN* system: specific instantiation of MBSC framework
- Key observation: row-wise attribute clusters (a-la fascicles) are not sufficient (e.g., \( Y = aX + b \))
- Idea: use carefully-selected collection of Classification and Regression Trees (CaRTs) to capture such “vertical” correlations and *predict values for entire columns*
### SPARTAN Example CaRT Models

<table>
<thead>
<tr>
<th>age</th>
<th>salary</th>
<th>assets</th>
<th>credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>30,000</td>
<td>25,000</td>
<td>poor</td>
</tr>
<tr>
<td>25</td>
<td>76,000</td>
<td>75,000</td>
<td>good</td>
</tr>
<tr>
<td>30</td>
<td>90,000</td>
<td>200,000</td>
<td>good</td>
</tr>
<tr>
<td>40</td>
<td>100,000</td>
<td>175,000</td>
<td>poor</td>
</tr>
<tr>
<td>50</td>
<td>110,000</td>
<td>250,000</td>
<td>good</td>
</tr>
<tr>
<td>60</td>
<td>50,000</td>
<td>150,000</td>
<td>good</td>
</tr>
<tr>
<td>70</td>
<td>35,000</td>
<td>125,000</td>
<td>poor</td>
</tr>
<tr>
<td>75</td>
<td>15,000</td>
<td>100,000</td>
<td>poor</td>
</tr>
</tbody>
</table>

- Can eliminate two data columns (predicted attributes) using a decision tree and a regression tree to

```plaintext
Can eliminate two data columns (predicted attributes) using a decision tree and a regression tree to
```
**SPARTAN Compression**

**Problem Formulation**

- **Given:**
  - Data table $T$ over set of attributes $X$ and per-attribute error tolerances

- **Find:**
  - Set of attributes $P$ to be predicted using CaRT models (and corresponding CaRTs+outliers) such that
    - Each CaRT uses only predictor attributes in $X-P$
    - Each attribute in $P$ is predicted within its specified tolerance
    - The overall storage cost is minimized
      - *materialization cost:* storage for predictor attributes in $X-P$
      - *prediction cost:* storage for CaRT models + outliers
SPARTAN Compression Problem

- Non-trivial problem!
  - Space of possible CaRT predictors is exponential in the number of attributes
  - CaRT construction is an expensive process (multiple passes over the data)
SPARTAN Architecture

From [S. Babu, M. N. Garofalakis, and R. Rastogi 01]
**SPARTAN’s DependencyFinder**

- **Input**: Random sample of input table T
- **Output**: A Bayesian Network (BN) identifying strong dependencies and “predictive correlations” among T’s attributes
- BN Semantics: An attribute is independent of all its non-descendants *given its parents*
- Use BN to restrict (huge!) search space of possible CaRT models: Build CaRTs using “neighboring” attributes (e.g., parents) as predictors
- *SPARTAN* uses an (enhanced) *constraint-based BN builder*
**SPARTAN’s CaRTSelector**

- “Heart” of the SPARTAN semantic-compression engine
- Uses the constructed Bayesian Network on T to drive the construction and selection of the “best” subset of CaRT predictors
- **Output:** Subset of attributes P to be predicted (within tolerance) and corresponding CaRTs
Complication: $A_n$ attribute in $P$ cannot be used as a predictor for other attributes
- Otherwise, errors will compound!!

Hard optimization problem — Strict generalization of Weighted Maximum Independent Set (WMIS) (NP-hard!!)

Two solutions
- Greedy heuristic
- Novel heuristic based on WMIS approximation algorithms
The CaRTBuilder Component

- **Input:** Random sample of the input table; target predicted attribute $X_p$; predictor attributes $\{X_1,\ldots,X_k\}$; and error tolerance for $X_p$

- **Output:** Minimum-storage-cost CaRT for $X_p$ using $\{X_1,\ldots,X_k\}$ as predictors, *within the specified error tolerance*

- **Contributions**
  - Novel algorithms for exploiting error tolerances in CaRT building
  - Integrated tree building and pruning for regression trees (dynamic programming algorithm)
The RowAggregator Component

- **Input**: Sub-table of materialized data attributes returned by the CaRTSelector
- **Output**: Fascicle-based (lossy) compression scheme for sub-table
- **Summary**
  - Attribute errors in sub-table should not propagate through the CaRTs to the predicted attributes
  - Algorithms based on fascicle algorithms [Jagadish, Madar, Ng 99]