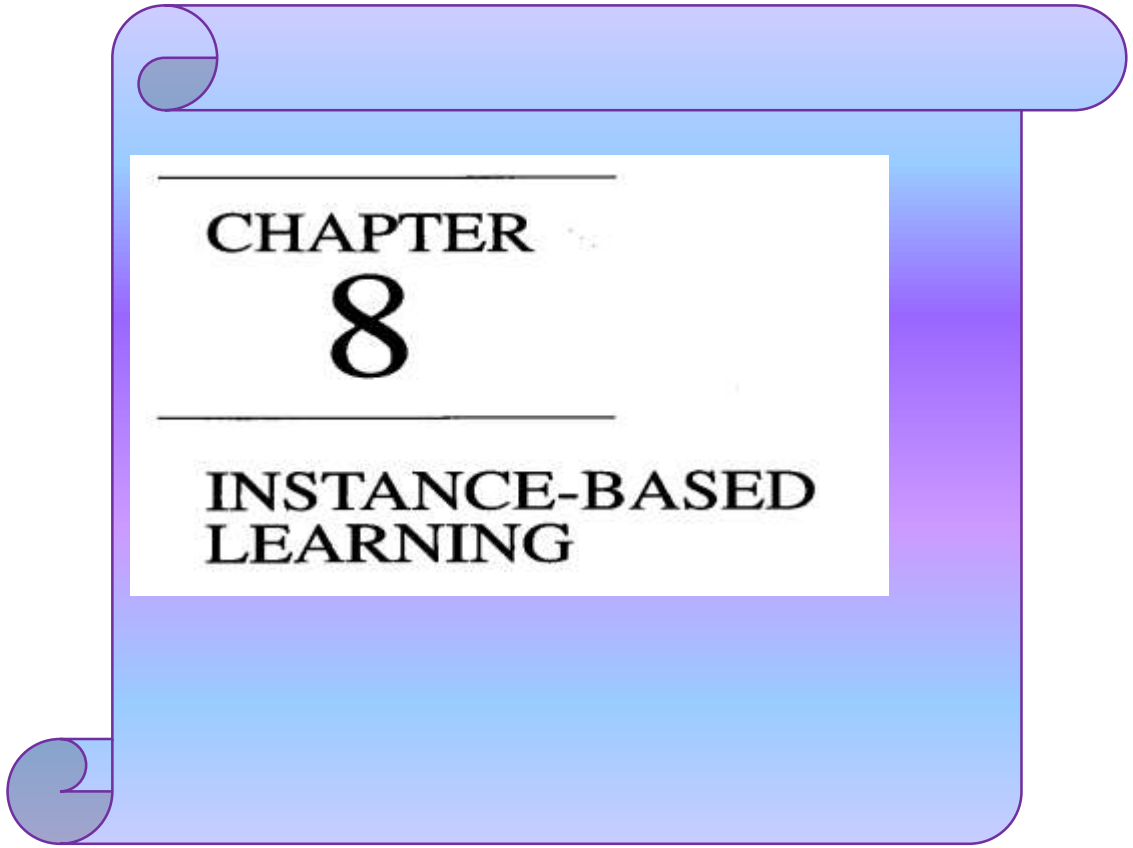
UNIT3 Part C Instance Based Learning



- Introduction
- K-Nearest Neighbor
- Locally Weighted Regression
- Radial Basis Functions
- Case-Based Reasoning
- Lazy and Eager Learning

B. Tech III Year
CS 701 PC

8.1 INSTANCE-BASE LEARNING

- Instance-based learning methods simply store the training examples instead of learning explicit description of the target function.
  - Generalizing the examples is postponed until a new instance must be classified.
  - When a new instance is encountered, its relationship to the stored examples is examined in order to assign a target function value for the new instance.
- Instance-based learning includes ***nearest neighbor*** , ***locally weighted regression*** and ***case-based reasoning*** methods.
- Instance-based methods are sometimes referred to as **lazy** learning methods because they delay processing until a new instance must be classified.
- A key advantage of lazy learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

- **Instance-Based Learning (Lazy Learning)**
  - Learning = storing all "training" instances
  - Classification = an instance gets a classification equal to the classification of the nearest instances to the instance

Instance/Memory-based Learning

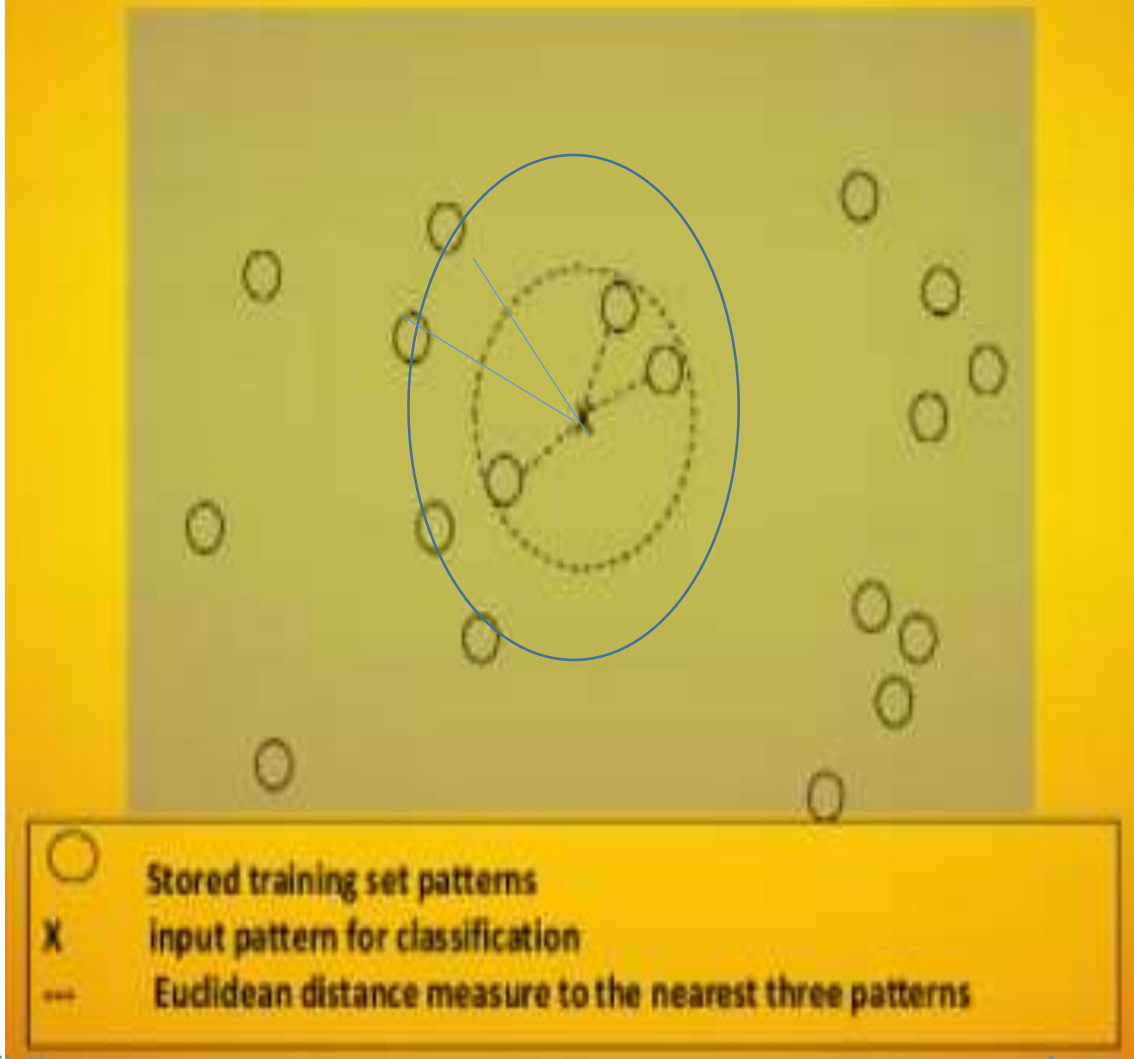
- **Non-parameteric**
  - Hypothesis(Assumption) complexity grows with the data
- **Memory-based learning**
  - Construct hypotheses directly from the training data itself

8.2 K Nearest Neighbors

- The key issues involved in training this model includes setting
  - **the variable K**
    - Validation techniques (ex. Cross validation)
  - **the type of distant metric**
    - Euclidean measure

$$Dist(X,Y) = \sqrt{\sum_{i=1}^{D} (Xi - Yi)^2}$$

Figure K Nearest Neighbors Example



k-Nearest Neighbor Learning Algorithm

- k-Nearest Neighbor Learning algorithm assumes all instances correspond to points in the n-dimensional space $\mathcal{R}^n$
- The nearest neighbors of an instance are defined in terms of Euclidean distance.
- Euclidean distance between the instances $x_i = \langle x_{i1}, \ldots, x_{in} \rangle$ and $x_j = \langle x_{j1}, \ldots, x_{jn} \rangle$ are:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (x_{ir} - x_{jr})^2}$$

- For a given query instance x_q, $f(x_q)$ is calculated the function values of k-nearest neighbor of x_q

The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

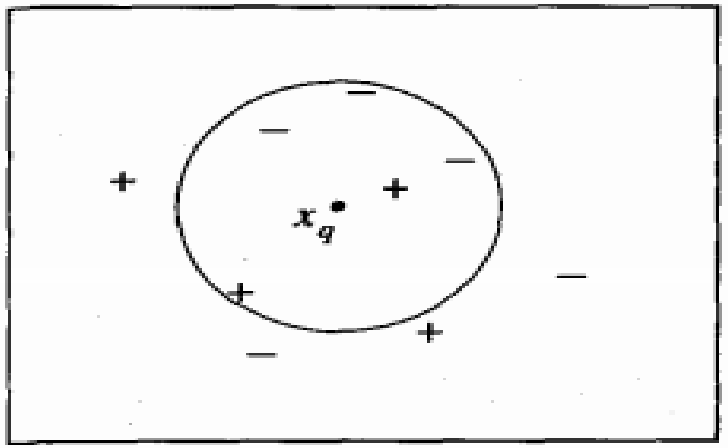
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

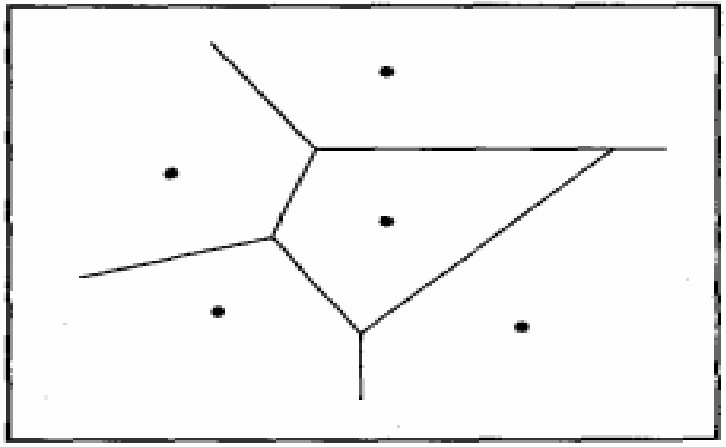
$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

---

k-NEAREST NEIGHBOR.
A set of positive and negative training examples is shown on the right, along with a query instance x q, to be classified.





Store all input data in the training set

↓

For each pattern in the test set

↓

Search for the K nearest patterns to the input pattern using a Euclidean distance measure

↓

For classification, compute the confidence for each class as C_i /K,

(where C_i is the number of patterns among the K nearest patterns belonging to class i.)

The classification for the input pattern is the class with the highest confidence.

When To Consider Nearest Neighbor

- Instances map to points in R^n
- Less than 20 attributes per instance
- Lots of training data
- **Advantages**
  - Training is very fast
  - Learn complex target functions
  - Can handle noisy data
  - Does not loose any information
- **Disadvantages**
  - Slow at query time
  - Easily fooled by irrelevant attributes

Training parameters and typical settings

- **Number of nearest neighbors**
  - The numbers of nearest neighbors (K) should be based on cross validation over a number of K setting.
  - When k=1 is a good baseline model to benchmark against.
  - A good rule-of-thumb numbers is k should be less than the square root of the total number of training patterns.

Training parameters and typical settings

- **Input compression**
  - Since KNN is very storage intensive, we may want to compress data patterns as a preprocessing step before classification.
  - Using input compression will result in slightly worse performance.
  - Sometimes using compression will improve performance because it performs automatic normalization of the data which can equalize the effect of each input in the Euclidean distance measure.

Issues

- **Distance measure**
  - **Most common: Euclidean**
  - Better distance measures: normalize each variable by standard deviation
  - For **discrete data, can use hamming distance**
- **Choosing k**
  - Increasing k reduces variance, increases bias
- For **"high-dimensional space", problem that the nearest neighbor may not be very close at all!**
- **Memory-based technique. Must make a pass through the data for each classification. This can be prohibitive for large data sets.**
- Indexing the data can help; for example KD trees

8.3 Distance-Weighted kNN Algorithm

Might want weight nearer neighbors more heavily...

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

and $d(x_q, x_i)$ is distance between x_q and x_i

Note now it makes sense to use *all* training examples instead of just k

Curse of Dimensionality

Imagine instances described by 20 attributes, but only 2 are relevant to target function

Curse of dimensionality: nearest nbr is easily mislead when high-dimensional X

One approach:

- Stretch jth axis by weight z_j, where $z_1, \ldots, z_n$ chosen to minimize prediction error
- Use cross-validation to automatically choose weights $z_1, \ldots, z_n$
- Note setting z_j to zero eliminates this dimension altogether

8.4 Locally Weighted Regression Algorithm

- KNN forms local approximation to f for each query point xq
- Why not form an explicit approximation f(x) for region surrounding xq

  ➔ Locally Weighted Regression
- **Locally weighted regression** uses nearby or distance-weighted training examples to form this local approximation to f.
- We might approximate the target function in the neighborhood surrounding x, using a linear function, a quadratic function, a multilayer neural network.
- The phrase "locally weighted regression" is called
  - ***local*** because the function is approximated based only on data near the query point,
  - ***weighted*** because the contribution of each training example is weighted by its distance from the query point, and
  - ***regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation f that fits the training examples in the neighborhood surrounding xq.
- This approximation is then used to calculate the value f(xq), which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

f is approximated near x_q using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \cdots + w_n a_n(x)$$

$a_i(x)$ denotes the value of the ith attribute of the instance x.

Minimize the squared error

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Kernel function K is the function of distance that is used to determine the weight of each training example.

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x)$$

Radial Basis Functions

Algorithm

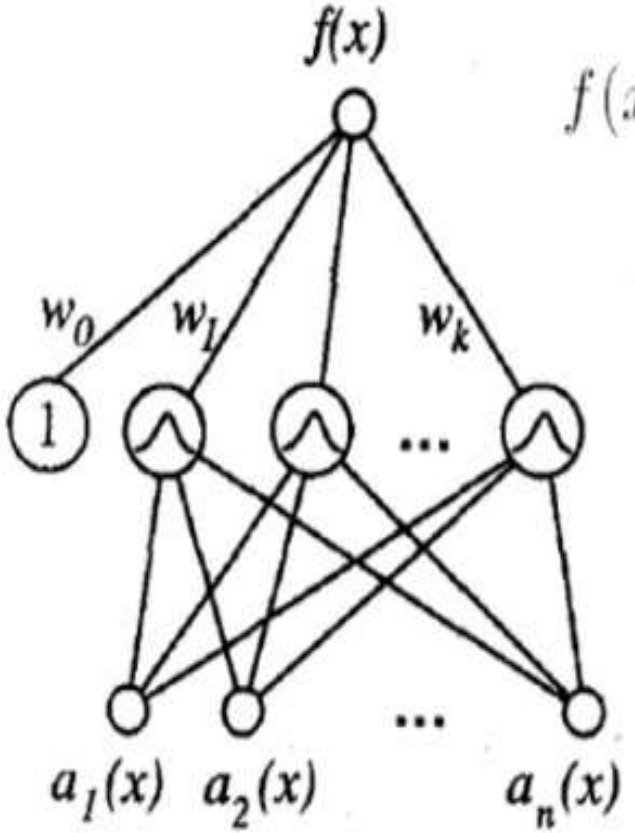
- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions.
- The learned hypothesis is a function of the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

where each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases. Here k is a user-provided constant that specifies the number of kernel functions to be included. Even though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u. It is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at the point x_u with some variance σ_u^2.

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Radial Basis Function Networks



$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Each hidden unit produces an activation determined by a Gaussian function centered at some instance *xu*.

Therefore, its activation will be close to zero unless the input x is near *xu*.

The output unit produces a linear combination of the hidden unit activations.

Training RBF Networks

- Stage one: define hidden units by choosing k, x_u and σ_u^2
  - Allocate Gaussian kernel function for each training example $\langle x_i, f(x_i) \rangle$.
  - Choose set of kernel functions that is smaller than the number of training examples.
    - Scatter uniformly over instance space
    - Or nonuniformly
- Stage two: train w_u
  - Gradient descent by global error criterion

Case-Based Reasoning

- Key properties of KNN and locally weighted regression:
  - Lazy learning
  - Analyzing similar instances
  - Points in Euclidean space
- For CBR:
  - Using more rich symbolic descriptions
  - Need different "distance" metric
  - Application:

    mechanical device design, legal cases reasoning…

- What is CADET? **Algorithm**
  - Employs CBR to design simple mechanical devices.
  - 75 stored examples of mechanical devices.
  - Training example:<qualitative function, mechanical structure>
  - New query: desired function
  - Target value: mechanical structure for this function

CADET System

prototypical eg. of a CBR system: **Assist in the conceptual design of simple mechanical devices such as water faucets.**

It uses a library containing approximately **75 previous designs** and **design fragments** to to **meet the specifications of new design problems**. Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.

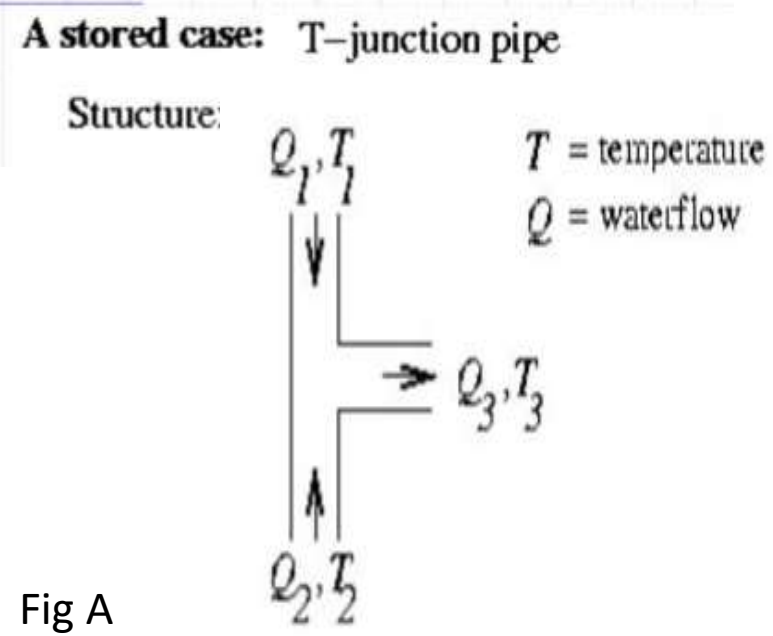


Fig A

This problem setting is illustrated in Figure a. The **left figure** shows the **description** of a typical stored case called a **T-junction pipe**. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs.

the **functional description** at its right, an arrow
th a **"+" label** indicates that the variable at the
owhead increases with the variable at its tail.

For example, the output **waterflow** Q3 increases with increasing input waterflow Ql. As shown in **b figure right**

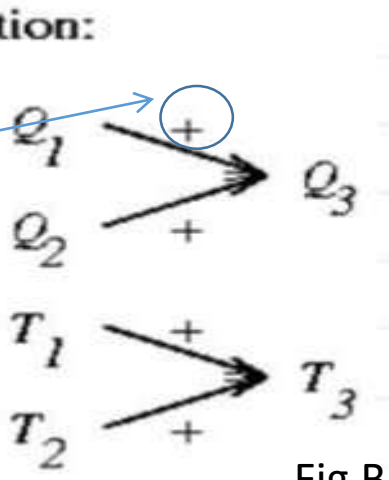
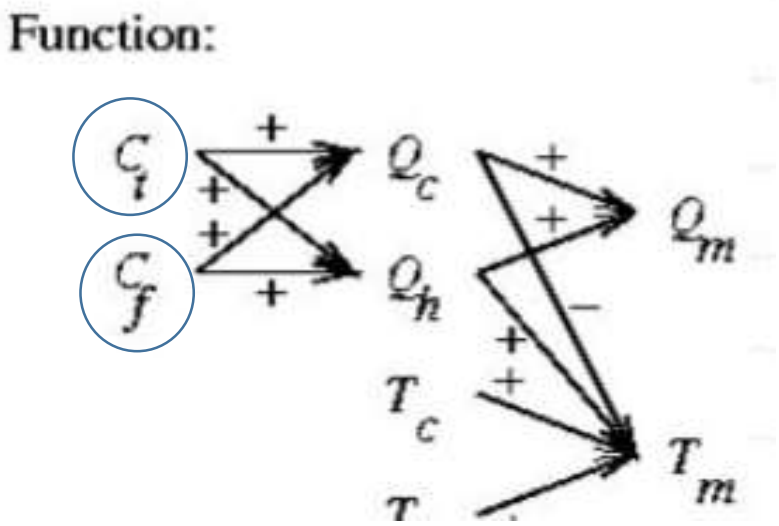


Fig B

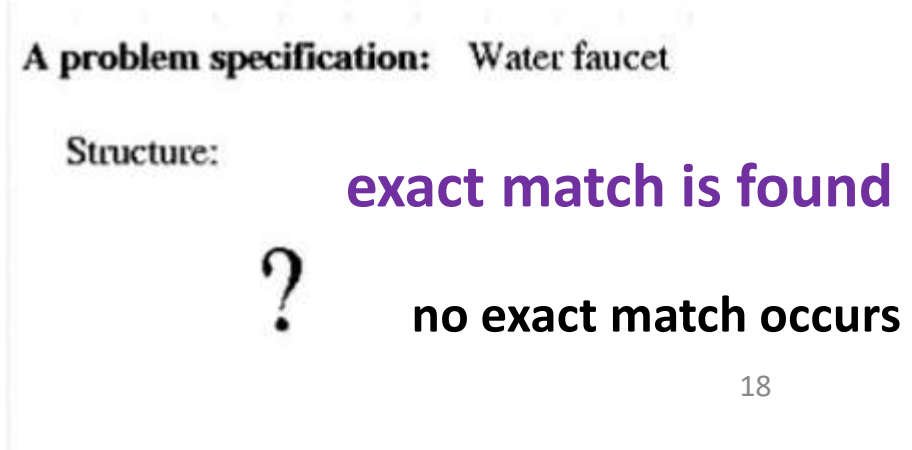
A stored case and a new problem. The above half of the figure describes a **typical design fragment in the case library of CADET**. The function is represented by the graph of qualitative dependencies among the T-junction variables (described in the text). The bottom half of the figure shows a typical design problem.



The variable C, denotes the control signal for temperature that is input to the faucet, and Cf denotes the control signal for waterflow. Note the description of the desired function specifies that these controls C, and Cf are to influence the water flows Q, and Qh, thereby indirectly influencing the faucet output flow Q, and temperature T,.

Given this functional specification for the new design problem, CADET searches its library for stored cases whose functional descriptions match the design problem.

****CADET searches for subgraph isomorphisms between the two function graphs, so that parts of a case can be found to match parts of the design specification.



Case-Based Reasoning in CADET

- Given function specification for new design, CADET search its library to find an exact match.
- If found, return this case.
- If not, find cases matching subgraphs. i.e., isomorphism subgraph searching, then piece them together.
- Elaborate the original function graph to match more cases.
  - eg: $A \xrightarrow{+} B$

  rewrite as: $A \xrightarrow{+} x \xrightarrow{+} B$

Correspondence between CADET and instance-based methods

- Instance space X: space of all function graphs
- Target function f: function graph -> structure
- Training example <x,f(x)>: describe some function graph x and structure f(x)

Several properties of CBR

- Instance represented by rich structural descriptions
  - CADET…
- Multiple cases retrieval (and combined) to form solution to new problem
  - KNN…
- Tight coupling between case retrieval, knowledge-based reasoning and problem solving.

Lazy and Eager Learning

- Lazy: wait for query before generalizing
  - KNN, locally weighted regression, CBR
- Eager: generalize before seeing query
  - RBF networks
- Differences:
  - Computation time
  - Global and local approximations to the target function
  - Use same H, lazy can represent more complex functions. (e.g., consider H=linear functions)