Model-Based Diagnosis

- Model = description of a system to be diagnosed
  - “objective” description.
  - usually component oriented.
  - may describe only correct behaviour or both correct and faulty behaviour.
- Diagnostic problem:
  - discrepancy between values predicted by the model and values observed in the real world
- Model-Based Diagnosis:
  - find source of discrepancy (component or fault)
  - by reasoning on the model.
**Temporal Decision Trees for Diagnosis**

Claudia Picardi
Università di Torino

**Decision Trees**

- A decision tree can be seen as a **classification procedure**.
- Classification problem:
  - $X$ belongs to a class among $C_1, \ldots, C_n$
  - You can carry out some tests $T_1, \ldots, T_k$ on $X$.
  - You must decide which class $X$ belongs to.

- A decision tree encodes a procedure to classify $X$.

**Example: Wine Classification**

- **Inner node**: a test $T$ to carry out on $X$.
- **Branches**: a subtree for each possible outcome of $T$.
- **Leaves**: final decision, i.e. a class $C$.

```
A1 Are you having exotic spicy food?
  a if yes: aromatic wine (e.g. gewurtztraminer)
  b if no: are you having red meat?
    c if yes: strong red wine (e.g. nebbiolo)
    s3 if no: are you having fish?
```
Building Decision Trees

Most common approach:
- **Learn** the decision tree from a set of examples
- Each example describes a situation where
  - The outcome of each test is known
  - The correct classification is known
- In other words, a table:

<table>
<thead>
<tr>
<th>Test T1</th>
<th>Test T2</th>
<th>...</th>
<th>Test T3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generic algorithm

Recursive function **BuildTree**

Input: Examples; Output: Tree

- if all examples belong to the same class **C**
  - return **C** as tree leaf
- else
  - **choose** a test **T** as root of the tree
  - for each possible outcome **v** of **T**
    - select only those examples where **T** has outcome **v**
    - recursively call **BuildTree** on them
  - the resulting tree becomes subtree of the current one.
- return the tree.
Choosing a test: the goal

- Goal: build the best decision tree.
- Best decision tree = tree with minimum average depth.

Why?
- Efficient decision process: Min. depth = Min. number of tests to carry out = Quicker classification.
- Generality: Avoid looking at unnecessary information that may depend on the particular set of examples.

ID3 [Quinlan, Machine Learning 1986]

- Choice based on the quantity of information carried by a test T.
  - Quantity of information = $-\text{entropy}$
- Measures the significance of a test T wrt the final classification:
  - If all classes are equiprobable wrt the outcome of T then entropy is max.
  - If exactly one class corresponds to each possible outcome of T then entropy is 0 (min).

Choose the test with min. entropy.
Temporal Decision Trees for Diagnosis
Claudia Picardi
Università di Torino

Recursive function **BuildTree**
Input: Examples; Output: Tree

- if all examples belong to the same class $C$
  - return $C$ as tree leaf
- else
  - choose the test $T$ with min. entropy as root
  - for each possible outcome $v$ of $T$
    - select only those examples where $T$ has outcome $v$
    - recursively call **BuildTree** on them
  - the resulting tree becomes subtree of the current one.
- return the tree.

On-Board Diagnosis

Two categories of diagnostic software:

- **Off-line**: Runs after the device has stopped operating. Its purpose is **repair**.
- **On-line**: Runs while the device is operating. Its purpose is **recovery**.

**real time** requirements
Two categories of diagnostic software:

**Off-line:**
Runs after the device has stopped operating. Its purpose is **repair**.

**On-line:**
Runs while the device is operating. Its purpose is **recovery**.

**On-board:** On-line +
Runs **embedded** within the device.

Current model-based engines are not suited to most on-board applications:
- they need **too much memory**
- they are **not swift enough**

They can however be exploited to...
- generate **specifications** for on-board software
- generate directly **on-board diagnostics**.

A possible approach:
- use an MBD engine to generate **decision trees**
Temporal Decision Trees for Diagnosis

Claudia Picardi
Università di Torino

Decision Trees for Diagnosis

- **Inner Nodes**: sensors to read
- **Children**: possible sensor values
- **Leaves**: recovery actions to perform

Generating Decision Trees

1. Exploit an MBD engine to generate a table:
   - the table associates combinations of sensor values to recovery actions.
   - can be built by simulating the device exhaustively for all relevant fault combinations.
   - or by diagnosing a significant set of real cases.
2. Use the table to generate a decision tree.
   - most used algorithm: **ID3**.
   - choose most **discriminating** sensor as a new node
   - **create a children** for each possible value
   - if a value allows to select an action create a **leaf**.
   - otherwise create an **inner node**.
   - proceed recursively.

---

**Model-Based Diagnostic Engine**
This approach does not exploit temporal information.

With a temporal MBD engine it is possible to generate a table with temporal information.

1. The tree cannot express temporal information. 
   temporal decision trees
This approach does not exploit temporal information.

2. the ID3 algorithm does not deal with temporal information. **Temporal ID3**

An example of correct diagnosis:
- Sensors: s1,...,sk
- Snapshots: 0,1,...,T
- for each pair (s,t) the value of s at time t.
- the correct recovery action to perform.

Additional information:
- Deadline for performing the action.
- Partial order defined on actions: recovery capability.
- Cost function defined on actions, must respect the partial order.
Temporal Decision Trees

| µinner nodes have also time labels
µwhen the tree is executed a counter is activated
µtime labels express the value the counter must have in order to read the sensor.

TDT Properties

* Time Labels must be increasing from root to leaves
  - The diagnostic software cannot go back in time to read previous sensor values
  - Sensor values cannot be stored (memory restrictions)
* The time label cannot exceed the deadline for any of the current candidates.
  - rather than exceeding the deadline perform a recovery action immediately
  - select the one with highest recovery capability (or combine several actions)
Why Waiting Is Good

- Decision Trees suggest a sequence of steps where the steps are either reading a sensor or performing a recovery action.
- Temporal Decision Trees introduce waiting.
- Loss of time? No.
  - avoid introducing unnecessary nodes in the tree
  - keep the tree smaller.
  - of course must guarantee to select a recovery action in time.

Why Not Using ID3?

- For Decision Trees: ID3 selects the most discriminating sensor.
- For Temporal Decision Trees: it could select the most discriminating sensor and time label.
- Problem:
  
  Moving ahead in time causes loss of information.
  It is not possible to go back and read previous sensor values.
**Why Not Using ID3?**

Moving ahead in time causes **loss of information**. It is not possible to go back and read previous sensor values.

---

**Measuring Information Loss**

- **Max discrimination degree**: given by example set.
  - Tree that uses all sensors at all snapshots = max discrimination degree.

- **Loss of information**:
  - The example set is reduced
  - The resulting discrimination degree could be lower.
  - Loss of discrimination capability.

- How can we **quantitatively measure** it?
Temporal Decision Trees for Diagnosis
Claudia Picardi
Università di Torino

Expected Cost of a TDT

- What happens when it is not possible to discriminate?
  - Choose the action with the highest recovery capability.
  - In some cases I perform a stronger action than needed.
  - Stronger actions cost more.
  - I pay a higher cost.
- Idea: measure discrimination capability with the **expected cost** of the tree.

Expected Cost Function

- Expected cost of a tree:

\[
X(n) = \begin{cases} 
\chi(\text{Act}(n)) & \text{if } n \text{ is a leaf} \\
\sum_{c \text{ child of } n} P(\mathbb{E}(c) \mid \mathbb{E}(n)) \cdot X(c) & \text{otherwise}
\end{cases}
\]

- Recursively computed from subtrees.
- Expresses the expected cost of a recovery action selected by the tree.
Properties of Expected Cost

1. It correctly measures discrimination capability:
   ✖ A more discriminating tree has a lower cost.

2. When I choose a pair (s,t) with higher t, it may decrease but it does not increase.
   ✖ It measures also information loss.

3. It depends only from the tree leaves
   ✖ It does not depend on the internal structure of the tree.

New Goal

✖ When generating the tree...
✖...choose a pair (s,t) of a sensor s and snapshot t such that:

1. The resulting tree has the minimum possible expected cost.

2. The resulting tree is reasonably small.
Temporal ID3 (sketch)

- When choosing a sensor $s$ and a snapshot $t$:
  - moving ahead in time I may increase the cost.

1. find the point in time where the loss becomes critical (the cost increases)
Temporal Decision Trees for Diagnosis

Claudia Picardi
Università di Torino

Temporal ID3 (sketch)

- When choosing a sensor \( s \) and a snapshot \( t \):
  - moving ahead in time I may increase the cost.
  1. find the point in time where the loss becomes critical (the cost increases)
  2. choose ID3-like a sensor and a time label, but only among safe time labels.

Temporal ID3 (sketch)

- Moving ahead in time I lose more and more information.
  1. find the point in time where the loss becomes critical (the cost increases)
  2. choose ID3-like a sensor and a time label, but only among safe time labels.
Temporal ID3

- How to determine whether the cost decreases or not for a given time label $t$?
  - build a lowest-cost decision tree using only information at times $\geq t$.

- **Problem**: if I build this tree recursively, I am performing exhaustive search!!!
  - It is like looking explicitly at all the possibilities and selecting the best one.
  - Not clever, not efficient.

First Step towards a Solution

- These trees are needed only for their cost.
  - The cost does not depend on the internal structure of the tree.
  - In a sense, the cost depends only on the example set used for building the tree.
  - These trees do not need to be small.
  - Why bother with recursive calls? Build the simplest tree.

- Build the **exhaustive** tree.
  - The tree using all sensors and snapshots remaining in the data set.
Second Step towards a Solution

- I must build $\text{TREE}(t_0), \text{TREE}(t_0+1), \ldots, \text{TREE}(T)$
  - $\text{TREE}(i)$ uses only information at times $\geq i$
  - $\text{TREE}(i-1)$ uses all the information of $\text{TREE}(i)$ and some more.
  - The cost does not depend on the inner structure of the tree.
- Build $\text{TREE}(i-1)$ by adding nodes to $\text{TREE}(i)$.
  - I can build all the exhaustive trees...
  - ...while I am building $\text{TREE}(t_0)$!
- **Same complexity as standard ID3**!
  - $O(N^2MT)$ in the worst case and $O(NMT/\log N)$ in the best case

An Extension

- Deadlines are **hard**.
  - A deadline either is met or not.
  - There are situations in which there is a tradeoff between the risk of waiting and the risk of doing the "wrong" action.
- Example: **preventive diagnosis**.
  - Waiting: risk that the fault occurs.
  - Not discriminating properly: risk to stop the device when it is not necessary.
Deadlines are **hard**.
- A deadline either is met or not.
- There are situations in which there is a tradeoff between the risk of waiting and the risk of doing the “wrong” action.

Introduce **soft deadlines**:
- There is an additional cost associated to waiting
- This additional cost depends also on the fault situation.

The new cost function

The cost of a leaf now depends:
- on the **action** it selects.
- on the **fault situations** it corresponds to.
- on the **time** it is reached at.

\[
W(n) = \begin{cases} 
\sum_{e \in \mathcal{E}(n)} P(e \mid \mathcal{E}(n)) \cdot [\gamma(\text{Act}(n), e) + \delta(e, \text{Time}(n))] & \text{if } n \text{ is a leaf} \\
\sum_{c \text{ child of } n} P(\mathcal{E}(c) \mid \mathcal{E}(n)) \cdot W(c) & \text{otherwise}
\end{cases}
\]
The Goal

- Build a **temporal decision tree**:
  - where sensor readings are used in the order they are read (same as in the original TID3).
  - that has the minimum cost (same as in the original TID3 but with a **different cost function**)
  - trying to keep the tree small in order to comply with the space constrain.

- Can we use TID3 as it is?
  - **No.**

Generating the Tree: a Problem

- Conditions on the **cost function** for TID3 to work:
  1. The cost of a tree must depend only on how the fault situations split among the tree leaves.
  2. A more discriminating tree must not have an higher cost than a less discriminating one.
- Function **W violates** both conditions!
  1. The cost depends also on the time at which tree leaves are reached.
  2. A more discriminating tree could have an higher cost due to the time at which it selects recovery actions.
**Temporal Decision Trees**

**Generating the Tree: Solution (outline)**

- **Idea:** *Transform* the problem with soft deadlines into a problem with hard deadlines so that:
  1. The solution (a tree) to the transformed problem is also a solution to the original problem.
  2. The transformation does not increase the complexity of the algorithm.
- We introduce a *preprocessing* step.

**Preprocessing 1: the Exhaustive Tree**

- First step: build the **exhaustive tree**.
  - Temporal Decision Tree that uses *all information* in the example set.
  - It is definitely the **most discriminating** tree.
  - It is **not the cheapest**.
  - It is not **small**.
- This step requires visiting once the whole example set:
  - **Complexity = O(NMT)**
  - N=#rows; M=#sensors; T=#snapshots
  - NMT = size (# of cells) of the example set.
Preprocessing 2: Pruning

- Visit the tree from the leaves to the root.
- For each inner node ask:
  
  Do I get a cheaper subtree by replacing this node with a leaf?

- NO: Don’t prune.
Temporal Decision Trees for Diagnosis
Claudia Picardi
Università di Torino

Preprocessing 2: Pruning

- Visit the tree from the leaves to the root.
- For each inner node ask:

  Do I get a cheaper subtree by replacing this node with a leaf?

- **NO**: Don’t prune.
- **YES**: Prune.
Visit the tree from the leaves to the root.
For each inner node ask:

**NO**: Don’t prune.
**YES**: Prune.

Do I get a cheaper subtree by replacing this node with a leaf?
Visit the tree from the leaves to the root.
For each inner node ask:

Do I get a cheaper subtree by replacing this node with a leaf?

- **NO**: Don’t prune.
- **YES**: Prune.
Preprocessing 2: Pruning

- Visit the tree from the leaves to the root.
- For each inner node ask:
  
  Do I get a cheaper subtree by replacing this node with a leaf?

- **NO**: Don’t prune.
- **YES**: Prune.

- Complexity: $O(NMT)$
**Properties of the Pruned Tree**

**Theorem:** let $T_p$ denote the pruned tree. For every other temporal decision tree $T$ for the same set of examples:

1) $W(T) \geq W(T_p)$

2) if there is a fault situation $e$ for which $T_p$ selects a recovery action at time $t$, while $T$ selects it at time $t' > t$, then $W(T) > W(T_p)$

1. The pruned tree has **the minimum possible cost** for the given set of examples.
2. Any tree exploiting information which has been pruned will **not** have minimum cost.

**Why is it so?**

[Proof Sketch: part 1]

- **The pruning step** selects one of the cheapest among all prunings.
  - It tries out all prune operations and carries out only those that lower the cost of the tree.
  - The obtained pruning has thus the lowest possible cost.
  - Moreover, less invasive prunings are all more expensive.
  - This shows that no unnecessary prunings are carried out.
- The **theorem** holds for prunings.
Given any temporal decision tree for an example set:
- we can complete it into a pruning with the same or lower cost.
- The completed tree selects recovery actions at the same time as the initial one.

This allows us to extend the result
- from the set of prunings...
- ...to the set of all temporal decision trees.

Transforming the problem:
- The pruned tree can be used to define hard deadlines.
- The hard deadline \((hd)\) for a fault situation is

\[ hd(e) = \text{Time(Leaf}(e)) \]
We have defined a **preprocessing step** that:

- **Transforms** the problem with *soft deadlines*, _not_ solvable by TID3, into a problem with *hard deadlines*, that TID3 can solve.
- Has a complexity of $O(NMT)$.
- Since the complexity of TID3 is $O(N^2MT)$ in the worst case and $O(NMT\log N)$ in the best case, the overall complexity is **unchanged**.