Derin Gezgin | Russell Kosovsky | Jay Nash
Fall 2024 | COM316: Artificial Intelligence
Homework 13: Case Based Reasoning

Create a CBR system that can reason about the passableness of objects that it has never seen. New objects encountered should be added to the data base to allow their use in future reasoning. New objects should have a full description of characteristics except for their passableness, which is what the CBR will determine. Use at least 2 objects stored in the database and the description of a new object to explain how your system works.

Our Case-Based Reasoning System

Our case-based reasoning system can start with two objects in its knowledge base:
Crate → ***Material****: Wood* | ***Height****: Medium* | ***Weight****: Heavy* | ***Color:*** *Red* | ***Passable:*** *Yes*
Wall → ***Material****: Brick* | ***Height****: Large* | ***Weight****: Very Heavy* | ***Color:*** *Blue* | ***Passable:*** *No*

As the first example, our robot can encounter with a Wooden Box with the following attributes:
Wooden Box → ***Material****: Wood* | ***Height****: Medium* | ***Weight****: Medium* | ***Color:*** *Blue*

In this case, our robot will **retrieve** examples from its knowledge base and look for the best match:
Crate → ***Material****: Match* | ***Height****: Match* | ***Weight****: No-Match* | ***Color:*** *No-Match*
Wall → ***Material****: No-Match* | ***Height****: No-Match* | ***Weight****: No-Match* | ***Color:*** *Match*

As the crate has more matches than the wall, our robot will **reuse** the crate example to get a passability which would be passable. When the robot tries to pass it will see (**revise)** that the wooden box is passable and **retain** the fact that the wooden box is passable to its knowledge base. This is what the knowledge base of the robot would look like:
Crate → ***Material****: Wood* | ***Height****: Medium* | ***Weight****: Heavy* | ***Color:*** *Red* | ***Passable:*** *Yes*
Wall → ***Material****: Brick* | ***Height****: Large* | ***Weight****: Very Heavy* | ***Color:*** *Blue* | ***Passable:*** *No*
Wooden Box → ***Material****: Wood* | ***Height****:Medium* | ***Weight****:Medium* | ***Color:*** *Blue* | ***Passable:****Yes*

We can do another negative example to see how our CBS system works:
Table → ***Material:*** *Wood* | ***Height:*** *Medium* | ***Weight:*** *Light* | ***Color:*** *Red*

In this case, our robot will **retrieve** examples from its knowledge base and look for the best match.

Crate→***Material****:Match*|***Height****:Match*|***Weight****:No-Match*|***Color:****Match*|***Passable:****Yes*
Wall→***Material****:No-Match*|***Height****:No-Match*|***Weight****:No-Match*|***Color:****No-Match*|***Passable:****No*
Wooden Box→***Material****:Match*|***Height****:Match*|***Weight****:No-Match*|***Color:****No-Match*|***Passable:****Yes*

In this case, we can see that the crate has the highest match count again. Our robot will **reuse** the crate example to conclude a table is passable. When our robot tries to

execute this decision, it will **revise** it as a table is not passable. Lastly, it will add that a table is not a passable object to its knowledge base:

Crate → ***Material****: Wood* | ***Height****: Medium* | ***Weight****: Heavy* | ***Color:*** *Red* | ***Passable:*** *Yes*
Wall → ***Material****: Brick* | ***Height****: Large* | ***Weight****: Very Heavy* | ***Color:*** *Blue* | ***Passable:*** *No*
Wooden Box → ***Material****:Wood* | ***Height****:Medium* | ***Weight****:Medium* | ***Color:****Blue* | ***Passable:****Yes*
Table → ***Material:*** *Wood* | ***Height:*** *Medium* | ***Weight:*** *Light* | ***Color:*** *Red* | ***Passable:*** *No*

How would you use indexes to improve this systems performance if it had several more objects in the data base?

From our example, we can see that the database can get big quickly, and iterating through each attribute can take a long time. This is not an efficient way to store the attributes. As an alternative, for each different attribute, we can store the indexes of the objects that has that attribute. For example, our final state from the previous part would look like this:

Material: *Wood* [1, 3, 4] | *Brick* [2]
Height: *Medium* [1, 3, 4] | *Large* [2]
Weight: *Heavy* [1] | *Very Heavy* [2] | *Medium* [3] | *Light* [4]
Color: *Red* [1, 4] | *Blue* [2, 3]

For the passability we would have a reverse format;
Passably: [1, 3]: *Passable* | [2, 4]: *Not Passable*

For example if we have a new object;
Curtain → ***Material:*** *Cotton* | ***Height:*** *Large* | ***Weight:*** *Light* | ***Color:*** *Green*

After this example our system would first filter out the attributes;
Material: *{Cotton is a new attribute}*
Height: *Large* [2]
Weight: *Light* [4]
Color: *{Green is a new attribute}*

As we only have one 2 and one 4, we can check both of their passability which is Not Passable. In other cases, we would check for the number we encounter the most. It will **revise** its decision as a curtain is passable. This is what our knowledge base would look like after this example:

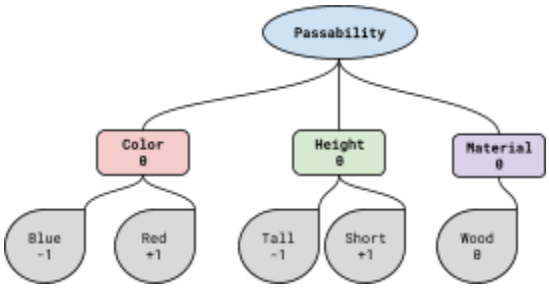
Material: *Wood* [1, 3, 4] | *Brick* [2] | *Cotton* [5]
Height: *Medium* [1, 3, 4] | *Large* [2, 5]
Weight: *Heavy* [1] | *Very Heavy* [2] | *Medium* [3] | *Light* [4, 5]
Color: *Red* [1, 4] | *Blue* [2, 3] | *Green* [5]
Passably: [1, 3, 5]: *Passable* | [2, 4]: *Not Passable*

Derin Gezgin | Russell Kosovsky | Jay Nash
Fall 2024 | COM316: Artificial Intelligence
Homework 13: Case Based Reasoning

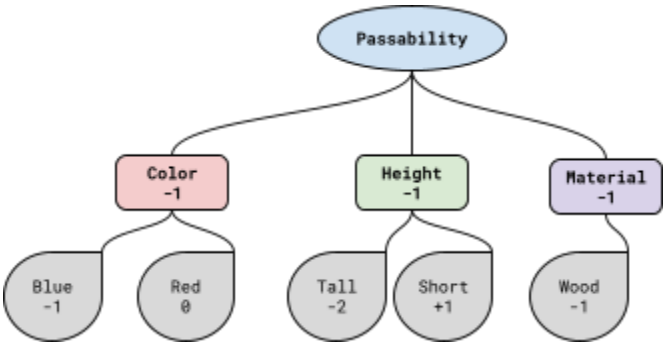
Consider also a CBR system that does not make a separate determination for passableness; it uses what it knows about it's current state and a database of past states to determine it's next move.
Address the issues concerned with retrieval, adaptation, and evaluation for each of the methods you describe.

In our modified case-based reasoning system, we can have a graph-like structure to represent our current knowledge base. To explain it from an example:

Initial State
We know that;
***Height**: Tall | **Color**: Blue | **Material**: Wood → Not Passable*
***Height**: Short | **Color**: Red | **Material**: Wood → Passable*
In this case, this is what our knowledge graph would look like

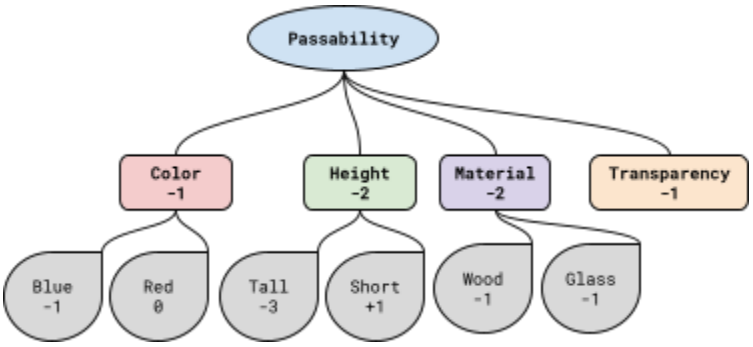


Example 1:
In this example, our robot encounters the following object:
***Height:** Tall | **Color:** Red | **Material:** Wood*
As our knowledge base is not large enough, the sum of the attributes in the graph is 0. This means that our robot will make a random decision. Regardless of its decision, it will learn that this combination is not passable and will update its graph like the following:



Example 2:
In this example, our robot encounters the following object:
Height: *Tall* | ***Transparency:*** *Transparent* | ***Material:*** *Glass*

Considering these attributes, our robot has only seen *tall* as before so it will take the direct score of tall (-2) and the section score of material (-1). This will not affect our passability score because we did not have a transparency section before. The total score for our passability is -3, so our robot will not try to pass. Our robot would confirm that this square is not passable and update its graph as the following:



Our method has a somewhat straightforward way of retrieval, but adapting to new information can be difficult because when we add a new branch (like transparency), the initial state is entirely predicated on a single sample. Without many samples, the robot will be very confident about that branch without a good reason to do so. Evaluation can also be difficult due to the fact that if we don't have most of the attributes already learned, we will have to rely on the branch sum values. In the later stages of the training, as our knowledge base gets bigger, we can be more sure about our decisions or guesses (about objects we have not seen), but this is not a very good solution for the early stages of the training.