AIML - Program AB Optimizations

Program AB Optimizations

Richard Wallace

Contact: info@alicebot.org
January 19, 2013

AIML Intermediate Format

An AIML bot typically consists of thousands, or hundreds of thousands, of AIML categories. Each category contains an input pattern and a template, and optionally a <that> pattern and a <topic> pattern. When we think of AIML content this way, it brings to mind row-oriented data that might be found in a database or spreadsheet. You can imagine AIML represented in a table, where the rows are the categories and the columns are labeled “pattern”, “that”, “topic” and “template”. What’s more, there are other attributes that can be associated with each category, for example the number of times each category is activated, and the name of the AIML source file containing the category. The imaginary spreadsheet might also include columns called “activation count” and “filename”.

Yet the AIML template is more like a hierarchical, tree-like structure. The simplest AIML template contains only text, but AIML allows the text to be marked up with AIML tags. These tags might enclose more text and more tags, giving rise to a structure like a tree. You can think of the template as the root of a tree, with branches leading to tags and text.

AIML is therefore a hybrid of database-style row data, and the hierarchical data in the templates. Because AIML is based on XML, and there is a significant amount of software written in a variety of languages for parsing XML, many AIML interpreters simply read the AIML files and process them as hierarchical data. An XML parser reads the AIML file, and parses it into a collection of nodes representing categories, and the nodes are further parsed into the component pattern, that, topic an template parts. The template is decomposed into its constituent hierarchical structure.

The problem with this approach is that, because of the size of the AIML files, the full XML parsing can be relatively slow. Also, there is no real need to parse all the individual templates until they are activated. For these reasons, we defined an intermediate AIML format that represents the categories as row data, and stores the templates as XML data. This format is called AIML Intermediate Format, or AIMLIF.

AIMLIF is a plain text format representing categories as line data.

One category per line, one line per category:

ACTIVATION_COUNT,PATTERN,THAT,TOPIC,TEMPLATE,FILENAME

In TEMPLATE, we replace “\n” with “#\Newline” and “,” with “#\Comma”, so that each category takes only one line of text.

Let’s look at an example. This AIML category comes from the file reductions.aiml:

```
<category>
  <pattern>DO YOU THINK YOU WILL *</pattern>
  <template><srai>WILL YOU <star/></srai>
  </template>
</category>
```

has an intermediate format representation

```
0,DO YOU THINK YOU WILL *,*,*,<srai>WILL YOU <star/></srai>,reductions.aiml
```

By preprocessing the AIML files into AIMLIF, our program can load the AIML much faster. Instead of parsing all the AIML XML files at load time, the program loads the AIMLIF files and stores the <template> XML for later use. The <template> XML is parsed only when the category is activated.

Unix Tools

A pleasant side-effect of using AIMLIF representation is that it facilitates applying common Unix tools to analyze and process the AIML. Because each category is represented as a single line of text, we can use tools like sed, awk, grep, sort and uniq on the AIMLIF files easily.

CSV File Format

The AIMLIF format is easily recognizable as the familiar spreadsheet .csv file format. AIMLIF files can be read and edited with spreadsheet tools, including MS Excel and Google Docs, facilitating such features as upload/download of AIML files in .csv format, and the development of customized spreadsheet editors for AIML.

Node Upgrades

Each node in the AIML graph is a map from words to successor nodes. Typically AIML interpreters use a hash table to represent the map. The current ALICE brain has 98,901 categories. The graph representing these categories has 613,051 nodes.

(The number of leaves is slightly less than the number of categories because the AIML contains some duplicate categoires).

```
Name = alice Path = c:/ab/bots/alice
Preprocessor: 395 norms 48 persons 9 person2
Loading AIML files from c:/ab/bots/alice/aimlif
Loaded 98901 categories in 2.372 sec
Bot alice 98901 completed 41 deleted.
613051 nodes 493215 singletons 98164 leaves 0 shortcuts 21672 n-ary 613050 branches
```

However the majority of nodes, more than 493,000, have only one branch. Here we call nodes with one branch “singletons” and nodes with more than one branch “n-ary”.

Rather than waste the overhead of a hash table on each node, when most have only one entry, we create each node as a singleton. When the program adds a new path to the graph, it may pass through an existing node and increase the number of branches. In that case, we “upgrade” the node by allocating a hash table.

Shortcuts

The AIML graph may be further compressed by taking advantage of the observation that, in a typical bot like ALICE or Mitsuku, most categories have only a <pattern>, and only a few have <that> and <topic> specified. Recall that the categories are represented by a path in the graph, from the root node to a leaf node. The <template> is stored in the leaf node. The general form of the path is:

PATTERN <THAT> THATPATTERN <TOPIC> TOPIC

A typical AIML category like:

```
<category>
<pattern>HOW ARE YOU</pattern>
<template>I am fine.</template>
</category>
```

may be printed as:

```
R--HOW-->N1--ARE-->N2--YOU-->N3--<THAT>-->N4--*-->N5--<TOPIC>-->N6--*-->”I am fine”
```

where R is the root node, and N1, N2...N5 are nodes in the graph. By far the majority of categories in our typical bots have this form, where the <that> and <topic> are set implicitly to the wildcard *. Subsequences like

```
N3--<THAT>-->N4--*-->N5--<TOPIC>-->N6--*-->
```

are repeated over and over in the graph.

The idea with a “shortcut” is to replace all the subpaths with <that> * <topic> * by a direct link to the matching template.

Here is a simple example with 3 AIML categories:

```
<aiml>
  <category>
    <pattern>*</pattern>
    <template>See you soon.</template>
  </category>
  <category>
    <pattern>YES</pattern>
    <that>SEE YOU SOON</that>
    <template>That context.</template>
  </category>
  <category>
    <pattern>YES</pattern>
    <template>No context.</template>
  </category>
</aiml>
```

The output of our program shows the difference between the graph without shortcuts, and the modified graph with shortcuts.

```
Name = small Path = c:/ab/bots/small
Preprocessor: 0 norms 0 persons 0 person2
Loading AIML files from c:/ab/bots/small/aimlif
Loaded 3 categories in 0.009 sec
Bot small 3 completed 0 deleted.
0(2)--YES-->14(1)--<THAT>-->15(2)--SEE-->16(1)--YOU-->17(1)--SOON-->18(1)--<TOPIC>-->19(1)--*--> That context....
0(2)--YES-->14(1)--<THAT>-->15(2)--*-->28(1)--<TOPIC>-->29(1)--*--> No context....
0(2)--*-->4(1)--<THAT>-->5(1)--*-->6(1)--<TOPIC>-->7(1)--*--> See you soon....
16 nodes 11 singletons 3 leaves 0 shortcuts 2 n-ary 15 branches 0.9375 average branching
```

The notation X(Y) means: Node X has Y branches. In this case the root node is 0 and it has 2 branches (“YES” and “*”), so it is written as 0(2).

The graph has 16 nodes consisting of 11 singleton nodes, 3 leaves and 2 n-ary nodes.

The same graph, with shortcuts:

```
Name = small Path = c:/ab/bots/small
Preprocessor: 0 norms 0 persons 0 person2
Loading AIML files from c:/ab/bots/small/aimlif
Loaded 3 categories in 0.002 sec
Bot small 3 completed 0 deleted.
34(2)--YES-->40(1)--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*-->No context....
34(2)--YES-->40(1)--<THAT>-->41(1)--SEE-->42(1)--YOU-->43(1)--SOON-->44(1)--<TOPIC>-->45(1)--*--> That context....
34(2)--*-->38(1)--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*-->See you soon....
9 nodes 7 singletons 1 leaves 2 shortcuts 1 n-ary 9 branches 1.0 average branching
```

The modified graph has replaced

```
--<THAT>-->5(1)--*-->6(1)--<TOPIC>-->7(1)--*--> See you soon....
```

with

```
--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*-->See you soon....
```

The X indicates that these are not “real” nodes, but rather a compressed representation of the path as a direct link form 38(1) to the template “See you soon.”

The graph compressed with shortcuts has 9 nodes comprising 7 singletons, 1 leaf, 2 shortcuts and 1 n-ary node. Two of the leaves from the original graph became shortcuts in the compressed graph. In the compressed graph, the sum of leaves plus shortcuts should equal the number of leaves in the original graph.

The results of shortcut compression with the ALICE brain are:

Without shortcuts:

```
Name = alice Path = c:/ab/bots/alice
Preprocessor: 395 norms 48 persons 9 person2
Loading AIML files from c:/ab/bots/alice/aimlif
Loaded 98901 categories in 2.372 sec
Bot alice 98901 completed 41 deleted.
613051 nodes 493215 singletons 98164 leaves 0 shortcuts 21672 n-ary 613050 branches
```

With shortcuts:

```
Name = alice Path = c:/ab/bots/alice
Preprocessor: 395 norms 48 persons 9 person2
Loading AIML files from c:/ab/bots/alice/aimlif
Loaded 98901 categories in 3.86 sec
Bot alice 98901 completed 41 deleted.
226969 nodes 203956 singletons 1565 leaves 96599 shortcuts 21448 n-ary 323290 branches
```

The total number of nodes is reduced by more than half, from about 613,000 to about 227,000.